Técnicas de estimación de rendimiento y área para el particionamiento hardware-software en un entorno de Codiseño.

Juan Antonio Maestro de la Cuerda

TESIS DOCTORAL

Universidad Complutense de Madrid
Escuela Superior de Informática
Dpto. de Arquitectura de Computadores y Automática

Técnicas de estimación de rendimiento y área para el particionamiento hardware-software en un entorno de Codiseño.

Memoria presentada por Juan Antonio Maestro de la Cuerda para optar al grado de Doctor en Informática por la Universidad Complutense de Madrid, realizada bajo la dirección de D. Daniel Mozos Muñoz.

Madrid, 22 de Abril de 1999

Este trabajo ha sido posible gracias a la Comisión Interministerial de Ciencia y Tecnología, por las ayudas recibidas a través de los proyectos CICYT TIC 94/0725-C03-02 y CICYT TIC 96/1071, y también gracias a la Comunidad Europea a través del proyecto HCM CHRX-CT94-0259.

Agradecimientos.

En primer lugar, quisiera agradecer a D. Daniel Mozos toda la ayuda y el apoyo vertidos en la tarea de dirección de esta tesis, así como su confianza a nivel personal y de trabajo, y las interesantes discusiones mantenidas durante todo este tiempo.

Asimismo, agradecer a D. Román Hermida los consejos que me ha ofrecido a lo largo de estos años, y todos los esfuerzos realizados para facilitar el desarrollo de mi trabajo.

También quiero agradecer a mis compañeros de *despacho*, D. Pablo Rupérez y D. José Manuel Mendías, así como a D. Julio Septién, a Dª. Hortensia Mecha y a Dª. Katzalin Olcoz, el agradable ambiente de trabajo que me han proporcionado, que sin duda ha contribuido a hacer posible mi *tarea de investigación*.

Además, recordar al resto de mis compañeros del Departamento de Arquitectura de Computadores y Automática, especialmente a D. Francisco Tirado y a Dª. Milagros Fernández, por la confianza que han depositado en mí en todo momento.

Por último, quiero agradecer a Dª. María Pilar Maestro el esfuerzo realizado en la siempre ardua tarea de la corrección ortográfica.

A Raquel, sin cuya ayuda esta tesis nunca se hubiera escrito.

A mis padres y a mi hermana, por todo el apoyo que me han ofrecido a lo largo de estos años.

Índice.

El Codiseño: Orígenes, estructura y trabajos previos.

1

1.1 Introducción.

La demanda actual de sistemas empotrados, compuestos de un procesador estándar que ejecuta un programa software, y uno o varios circuitos hardware de propósito específico (ASIC) crece rápidamente día a día, adaptándose a las exigentes necesidades del mercado. Esto se debe a la naturaleza de los productos actualmente solicitados, los cuales evolucionan de una manera progresiva hacia una mayor automatización de sus funciones [GuBS93] [WoFr92] [Wolf94].

Hoy en día, se utilizan en todos los campos dispositivos empotrados electrónicos, cuya complejidad se incrementa a medida que la aplicación a la que van destinados se mejora con un mayor número de operaciones. Raro es encontrar un sector de la sociedad en la que no intervengan elementos formados por circuitos digitales, junto con un programa software que los controle, por lo que estos componentes se han convertido en básicos para la realización de tareas esenciales.

Por una parte, existen algoritmos de control que tradicionalmente se han implementado en software, y que debido a su complejidad actual no pueden cumplir las restricciones temporales para las que fueron ideados. La

solución todavía adoptada en muchos casos, trivial por otra parte, de aumentar la potencia de cálculo del procesador asociado, deja de ser eficaz a medida que aumentan las exigencias de las aplicaciones, por lo que sería necesario añadir al sistema un componente hardware que fuese capaz de acelerarlo.

Por otra parte, hay circuitos hardware fabricados en serie que deben ser abaratados a toda costa, debido a la evolución actual del mercado. En consecuencia, deberían ser diseñados parcialmente en software, ya que de lo contrario podrían fracasar comercialmente frente a productos competidores más asequibles.

De esta manera, factores difíciles de conjugar *a priori*, como el rendimiento y el coste, resultan clave en el proceso de diseño. Por lo tanto, la producción rápida y eficiente de este tipo de elementos se ha convertido en una labor fundamental dentro de un mercado cada vez más competitivo.

Así, parece lógico tender hacia una automatización en el desarrollo de los sistemas empotrados, de la misma manera que se ha venido haciendo hasta ahora en otros campos, como el de la fabricación de circuitos integrados. Con esto se conseguirían dos metas importantes:

- Lanzar el producto en el menor tiempo posible, cumpliendo de esta manera las restricciones impuestas por una política de competencia dentro del mercado.
- Facilitar la realización de tareas tediosas, tradicionalmente realizadas por el diseñador, el cual queda libre para llevar a cabo otros procesos más importantes dentro de la metodología general.

Evidentemente, esta automatización no es una labor sencilla, ya que es difícil sustituir la experiencia previa de los diseñadores en este tipo de tareas, la cual resulta imprescindible a la hora de tomar ciertas decisiones trascendentes en las etapas de desarrollo. Esto hace que a menudo sea conveniente la interacción del ser humano dentro del proceso, para poder

dotarlo de un grado de conocimiento que de otro modo sería imposible alcanzar. Sin embargo, se tiende a intentar conseguir el mayor nivel de automatización posible.

Este hecho trae consigo un problema asociado a la naturaleza de los sistemas empotrados, y no es otro que el tratarse de sistemas híbridos. Estos sistemas están esencialmente formados por un componente hardware y uno software, junto con un interfaz que hace posible las comunicaciones entre ambos. Tradicionalmente, se ha venido realizando diseño automático de ambas partes por separado, mediante métodos largamente contrastados. Pero estas técnicas clásicas pierden su eficacia, debido a que se tienen que conjugar características de elementos muy distintos entre sí, labor esta en la que se carece de una experiencia probada.

De esta manera, y teniendo en cuenta todos los comentarios vertidos hasta ahora, se puede definir el *Codiseño* [BeLR97] [Erns98] [GVNG94] [StWo97] como *el conjunto de pasos destinados a la realización de un diseño híbrido, formado por un componente software y un componente hardware, de una manera automática.*

El hecho de disponer de elementos heterogéneos aumenta considerablemente el espacio de soluciones disponible. Esto es una gran ventaja, ya que en general los sistemas empotrados tienen altas restricciones, ya sean temporales o de coste que han de cumplirse obligatoriamente.

Así, mediante la técnica de Codiseño es posible alcanzar esta meta de una forma eficiente, conjugando adecuadamente las facilidades ofrecidas por los dos subsistemas de hardware y software. Estudiando las características de ambos componentes, se observa que el software tiene un coste reducido, pero su tiempo de ejecución es considerable, mientras que por otra parte, el hardware es mucho más rápido, pero su coste es también muy elevado. Esto obliga a trabajar con parámetros característicos

totalmente opuestos a la hora de llevar a cabo un diseño que cumpla los requerimientos exigidos, y a la vez ofrece más alternativas para combinar todos los elementos disponibles.

De esta manera, es posible acelerar sistemas que son demasiado lentos en software mediante un paso de sus funcionalidades más críticas a hardware, y abaratar sistemas excesivamente costosos en hardware mediante su transformación parcial a software.

Se puede observar que una solución híbrida como la que se propone modifica substancialmente la *arquitectura objetivo* del diseño final, sobre la que se realizan una serie de innovaciones. Así, a la hora de fijar estos cambios, habrá que considerar, entre otros, factores como los siguientes:

- Características del procesador que ejecutará el software, con la posibilidad de que sea segmentado.
- Características del sistema de memoria donde residirán el programa y los datos, con posibilidad de inclusión de memoria cache.
- Tipo del bus y de los interfaces que permitan la comunicación de datos entre el procesador y el ASIC.
- Posibilidad de comunicación mediante memoria compartida o paso de parámetros.
- Posibilidad de dotar al ASIC de acceso directo a memoria y de lógica para arbitrar el bus.

1.2 Estructura del proceso de Codiseño.

Una vez decidida la arquitectura objetivo, se comienza con las etapas propiamente dichas que conforman el proceso del Codiseño. En la Figura 1.1 se puede observar un esquema general, en forma de tareas interconectadas, de esta metodología.

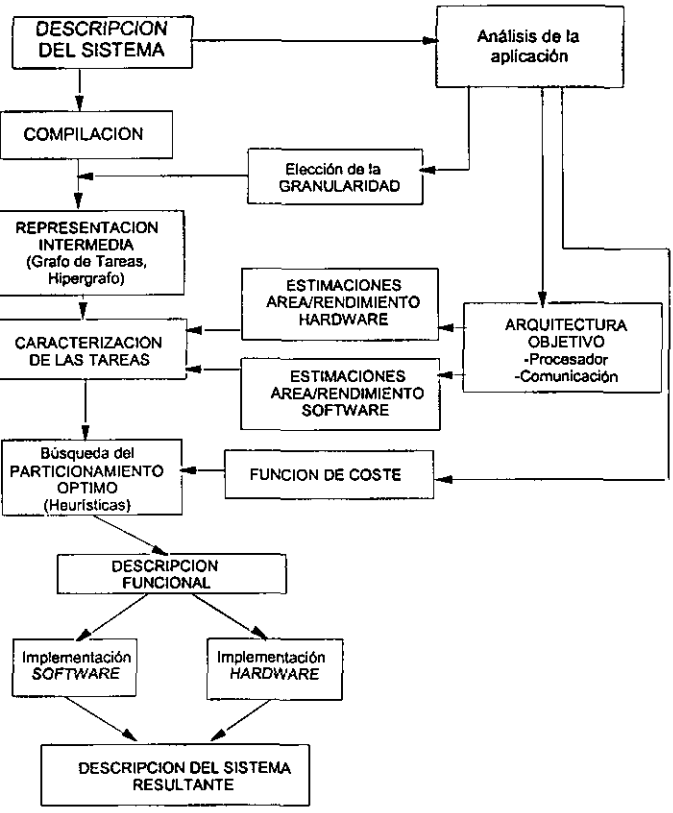


Figura 1.1

Estructura del proceso de Codiseño

Básicamente, los procesos que conforman el núcleo general se pueden enumerar de la siguiente manera:

1. Estudio de la especificación inicial y paso a la estructura interna mediante un proceso de compilación.
2. Aplicación de estimadores para la obtención de los parámetros de tiempo y coste característicos de las tareas individuales y del sistema global.
3. Proceso de particionamiento basado en los valores hallados en el punto anterior y en la definición de una función de coste apropiada.

4. Revisión de los resultados obtenidos y repetición del proceso en el caso de que no hayan sido los adecuados, con la posterior generación del software y del hardware ya particionados.

A continuación se exponen cada uno de estos puntos de una manera más desarrollada.

1.2.1. Proceso de compilación.

En el paso inicial, se parte de una especificación del sistema que puede estar representada en alguno de los lenguajes tradicionalmente usados. En esta especificación han de figurar todas las características necesarias para diseñar dicho sistema, así como las posibles restricciones de tiempo y de coste que hayan sido impuestas [AFSS98] [KuGa92].

Quizás uno de los mayores problemas en este punto esté relacionado con el lenguaje de especificación inicial [ViVe98] [VPGS98], al no existir uno que combine los tipos de descripción software con elementos esenciales del hardware, como la expresión de tiempos.

De esta manera, existen dos posibles orientaciones válidas a la hora de explicar el comportamiento del modelo, siendo la decisión de cuál resulta más conveniente dependiente de cada caso en concreto.

Así, descripciones que inicialmente son más representativas en software vendrán dadas en lenguajes como C [GhKL99] [MGKP97] o Ada [AVVH98] [LVSV97], mientras que aquellas que se adapten mejor al hardware, lo harán en VHDL [ElPD94] o Hardware-C [GuDe97].

Por ejemplo, si se tiene una aplicación específica de control, con unas altas restricciones temporales que cumplir, se partirá de una descripción software y a partir de ahí se migrarán funcionalidades a hardware. Pero si se dispone de un circuito integrado, se partirá de una descripción orientada a hardware y posteriormente se trasladará parte de sus elementos a software con el objetivo de reducir su coste.

Para que la especificación pueda ser manipulada por un sistema automático, es necesario compilarla a una estructura interna representativa sobre la que se puedan realizar las operaciones básicas de Codiseño de una manera sencilla. Realmente, todas las estructuras suelen tener la forma de un grafo, al cual se le dota de más o menos propiedades según sea necesario para las tareas que se van a desarrollar [DaJh98] [GrMa98] [GVNG94].

El objetivo que se busca en esta etapa es dividir la descripción inicial en trozos de código más pequeños, y asignarlos a distintos nodos del grafo. Los arcos con los que se interconectan representarán la existencia de una comunicación, o dependencia de datos, entre las distintas tareas relacionadas.

Un punto previo importante que hay que considerar es el tamaño de dichos nodos, lo que se conoce con el nombre de *granularidad*. De esta manera, si se dispone de un conjunto de tareas, las posibilidades de dividirlas en distintos bloques son muy diversas. No es nada obvio cuál de estas posibilidades es la que se debería escoger, ya que su elección influye decisivamente en los resultados finales.

Si se elige una granularidad fina [HeEr97], entonces se manejarán un gran número de bloques pequeños, estando expresada dentro de cada uno de ellos una reducida fracción del código inicial. Esto puede facilitar el estudio de sus características de tiempo y coste, debido a la simplicidad de cada nodo, pero dificulta las demás tareas de Codiseño, al aumentar el número de elementos a tratar.

Por otra parte, una granularidad gruesa [LCLS96] implica trabajar con un número pequeño de bloques más grandes, conteniendo cada uno de ellos extensas zonas de la descripción inicial. De esta manera, se simplifican en gran medida las comunicaciones, pero se tiene el efecto negativo de hacer más difícil la caracterización de los bloques individuales.

El hecho de escoger una granularidad apropiada incide en unos mejores y más rápidos resultados una vez finalizado el proceso de Codiseño. Sin embargo, esta elección es dependiente de cada caso, por lo que ha de ser tomada mediante un mecanismo de guía basado en un conjunto de heurísticas. Si cuando se haya completado el proceso global de Codiseño no se cumplen las expectativas originalmente propuestas, habrá que reinicializarlo con unas condiciones iniciales distintas, entra las que se debería encontrar la elección de la granularidad.

1.2.2. Estimaciones de los parámetros hardware y software.

Una vez que se tiene el equivalente a la especificación inicial compilada dentro de la estructura interna, hay que llevar a cabo lo que se conoce como *estimación* [GVNG94] [MaMM98] [VaGa95b]. El primer proceso de estimación, que es el más básico, consiste en asignar unos valores característicos a cada uno de los bloques o nodos que conforman la estructura interna.

Evidentemente, a la hora de tomar decisiones acerca de cómo se van a dividir los componentes entre hardware y software hay que basarse en alguna información que permita guiar al sistema hacia la mejor solución posible. Es decir, a partir de las características de cada uno de los bloques individuales se tiene que poder alcanzar un diseño que cumpla las propiedades impuestas. Si se tiene en cuenta la naturaleza de los sistemas codiseñados, es claro que los principales valores que hay que calcular son el tiempo y coste de cada nodo, tanto en su implementación software como en sus múltiples implementaciones hardware[1], aunque también sería posible estudiar otros factores, como las facilidades de test en el sistema [AILR96].

[1] Implementaciones estas que deben ser seleccionadas por el diseñador de entre el número indefinidamente alto de las posibles.

La dificultad que lleva implícita el proceso de estimación viene dada por el hecho de que se tiene que efectuar *a priori*, en una etapa muy temprana del proceso de Codiseño. Esto, unido a la posibilidad de no disponer de la arquitectura objetivo sobre la que se va a ejecutar el sistema final, hace que pueda resultar inviable la realización de medidas.

La diferencia entre medición y estimación reside en que los valores hallados mediante mediciones son exactos, mientras que los obtenidos por las estimaciones son aproximaciones del valor real, que se ven modificados por un margen de error el cual hay que disminuir lo máximo posible.

Existen varias formas de abordar estas estimaciones, cada una de ellas con sus ventajas y sus inconvenientes:

Por una parte, las estimaciones software [GuDe97] [LiMW95] se pueden llevar a cabo directamente sobre el código inicial en alto nivel de la especificación. Con esto se obtiene un ahorro de tiempo debido a que dicho código está disponible desde un principio, y no es necesario transformarlo mediante ningún proceso. Sin embargo, los valores resultantes pueden tener un margen de error elevado, debido fundamentalmente a la no consideración de los efectos producidos por el compilador.

También es posible realizar una etapa intermedia de compilado para solucionar el inconveniente anterior y obtener de una forma directa el código ensamblador. Las estimaciones producidas a partir de dicho código son más fiables, ya que se cuenta con un grado de detalle más alto. No obstante, esto puede acarrear dos problemas:

- Se emplea más tiempo en finalizar el proceso, debido a la etapa de compilación.
- No se pueden considerar las mejoras que realizará el compilador cuando genere el código procedente de la consideración de varios bloques simultáneamente.

Otro factor que influye ampliamente en las estimaciones es el de la arquitectura objetivo que se haya elegido. La disposición final de los componentes hardware y software que se van a diseñar variará según las características de la elección realizada. Dependiendo de si se considera memoria cache o de si el procesador está segmentado o no, el grado de dificultad para calcular las estimaciones será distinto. Evidentemente, si se dispone de alguno de los elementos anteriores los efectos colaterales producidos serán más difíciles de predecir, por lo que los resultados tendrán más índice de error.

De la misma manera, existe el problema de la indeterminación de los datos. Hay determinados valores que son desconocidos hasta que el sistema se comienza a ejecutar, ya que pueden generarse como consecuencia de la interacción humana o de algún fenómeno físico exterior. Para predecir el comportamiento de esos datos, no existe otra solución que realizar un estudio o perfil de los valores característicos con los que va a trabajar la aplicación, y así dar unos resultados posibles tras un análisis de probabilidades [MaML97] [WuLa94].

Sea como fuere, el objetivo consiste en relacionar un bloque de especificación, bien sea en alto nivel o en ensamblador, con el tiempo y coste necesarios para su implementación en software. Cuanto más ajustada sea esta relación al valor verdadero, las estimaciones gozarán de una mayor calidad.

A la hora de realizar las estimaciones hardware [MFTS96] [PoWo95] [VaGa95a] aparece el mismo problema: dado un bloque de especificación, hay que calcular el tiempo y coste necesarios para su implementación en este subsistema. De igual manera, este proceso se puede llevar a cabo a partir de un código en alto nivel (de comportamiento) o en bajo nivel (estructural), según se crea más conveniente.

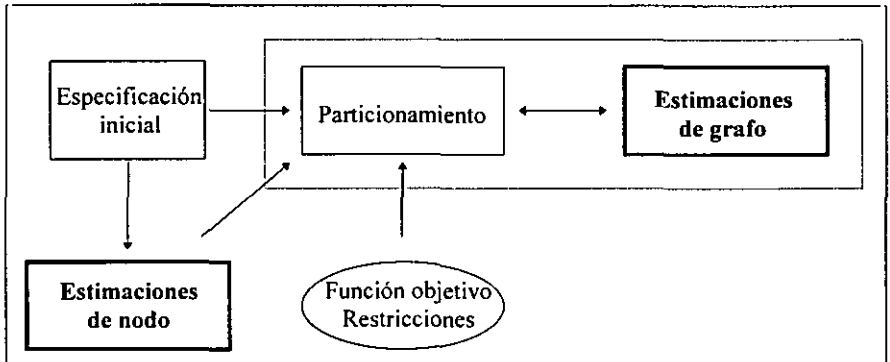


Figura 1.2

Estimaciones de nodo y estimaciones de grafo

Sin embargo, el hardware posee más dificultades de las que tenía asociadas el software. Esto se debe a que es preciso hacer frente a conceptos más complejos como la reutilización de módulos y el paralelismo entre tareas. Si, por ejemplo, se tienen dos bloques que desempeñan la misma función y que carecen de dependencias de datos, se puede optar por generar un único módulo físico, con lo que se obtendría un coste bajo con el inconveniente de incrementar el tiempo de ejecución, o bien generar dos módulos, con lo que se produciría una ejecución en paralelo, ahorrando tiempo e incrementando el coste. Estas alternativas dan lugar a la posibilidad de múltiples implementaciones hardware.

Todas estas decisiones son difíciles de tomar, sobre todo en fases tempranas del diseño, por lo que una alternativa válida consiste en hacer uso de alguna herramienta de síntesis de alto nivel. Gracias a esto, es posible predecir de una manera eficiente tanto el tiempo como el coste que se va a obtener con un diseño en particular, explorando rápidamente las distintas alternativas que se plantean.

Sin embargo, estas estimaciones iniciales, denominadas *estimaciones de nodo*, porque afectan a cada bloque de una manera individual, constituyen un proceso relativamente sencillo, comparadas con la

estimación de parámetros relacionados con todo el grafo de una manera global, lo que se denomina *estimaciones de grafo* (Figura 1.2).

Efectivamente, si se considera cada uno de los nodos por separado en un estadio inicial, la caracterización de sus parámetros asociados no se ve afectada por ninguna influencia proveniente del resto del sistema. De esta manera, y aunque la previsión de datos desconocidos pueda añadir cierta complejidad, los valores obtenidos deberían ser bastante fiables.

No obstante, los nodos se agruparán alrededor de dos particiones, la hardware y la software, y para generarlas, no se respetará la estructura interna de los nodos individuales, sino que se optará por aquella reestructuración que conlleve la obtención de un mejor resultado. De esta manera, la calidad del sistema final, que es lo que realmente interesa conocer, no es fácilmente derivable de los parámetros asociados a los nodos individuales.

Es precisamente este índice de calidad el que guiará al particionador a través del espacio de diseño, y de su precisión depende el éxito en la consecución de esta tarea. Si las estimaciones de nodo se han realizado con un elevado índice de error, o bien este error se ha introducido en la estimación global del grafo, el resultado final que se obtenga puede que no cumpla con las restricciones que se han impuesto al sistema y habría que repetir de nuevo todo el proceso [HeHE94]. Esto demuestra la importancia de disponer de unas buenas estimaciones, lo cual permite ahorrar tiempo en alcanzar un resultado aceptable.

Otro punto trascendente reside en la importancia de realizar las estimaciones en un periodo razonablemente corto. Es evidente que los resultados obtenidos pueden aproximarse mucho al valor real si se emplea un tiempo lo suficientemente grande, pero esto no interesa, ya que una de las metas que se tratan de conseguir es acortar al máximo el tiempo de diseño. Por lo tanto, hay que llegar a un compromiso entre la calidad de las

estimaciones y el periodo de tiempo que se necesita para conseguirlas, el cual puede tener una alta repercusión si se tiene en cuenta el gran número de veces que se realiza este proceso.

Además de las estimaciones hardware y software, es preciso tener en cuenta la interacción entre ambas particiones. De esta manera, al estar las tareas relacionadas, se tendrá asociado un intercambio de información entre nodos adyacentes, que, si se encuentran asignados a particiones distintas, producirá un consumo de tiempo por utilización del bus.

Este tiempo asociado a las comunicaciones tiene la característica de que depende de la situación relativa de los nodos entre sí, como es obvio. Por lo tanto, sus características irán modificándose según avanza el proceso de desarrollo y se tomen decisiones acerca de la asignación final de cada bloque. La fase de comunicación entre nodos de particiones distintas, que será proporcional al número de variables intercambiadas, constituirá un objetivo prioritario de minimización en el sistema.

Las decisiones que se tomen acerca de la composición de la arquitectura objetivo son fundamentales, ya que si, por ejemplo, se decide utilizar un ancho de bus elevado, se reduce el tiempo empleado en la comunicación de parámetros, pero se aumenta el coste inicial del sistema. Por lo tanto, hay que estudiar individualmente cada caso y tomar las opciones que se consideren más convenientes.

1.2.3. Proceso de particionamiento.

Tras haber realizado las estimaciones de nodos iniciales mencionadas anteriormente, se llega a una etapa fundamental dentro del proceso de Codiseño, que es la que se conoce como *particionamiento* [MaMo96] [VaGG93] [Vahi91]. En este punto, es necesario decidir cuáles de los bloques disponibles se implementan en software y cuáles en hardware, creando para ello dos particiones disjuntas.

Esta tarea resulta crítica y de ella dependerá que la solución final se acerque más o menos al grado de optimización que se pretende.

La meta principal es hacer que se cumplan todas las restricciones impuestas, ya sean totales, que afecten a todo el sistema, o parciales, que sólo tengan influencia en partes concretas [PoWo95].

Está claro que si se tiene una especificación software que no cumple una restricción temporal, se puede solucionar fácilmente migrando un número lo suficientemente grande de módulos a hardware. No obstante, esto encarecería inevitablemente el coste final, algo no deseado, por lo que este tipo de decisiones han de ser meditadas cuidadosamente. Hay que darse cuenta, que lo que interesa es sencillamente que la restricción temporal se cumpla, por lo que una vez logrado, es necesario una optimización del incremento de coste introducido en el sistema.

Por lo tanto, lo que se pretende es una migración de bloques reducida pero suficiente entre las dos particiones. Sin embargo, como cada uno de estos bloques tiene sus características intrínsecas, no sólo es necesario saber cuántos, sino cuáles de ellos son los que se trasladan.

Este problema se ve agravado por el factor comentado previamente, de que el hecho de migrar un nodo entre dos particiones supone una modificación en las comunicaciones con respecto a todos los nodos adyacentes. Así que es necesario evaluar en cada momento el estado global del sistema para evitar que una decisión aparentemente buena en un principio conduzca al proceso hacia regiones no adecuadas del espacio de búsqueda.

También hay que destacar el hecho de que el particionamiento se realiza en función de los valores estimados anteriormente para cada nodo, de una manera individual. Así, se utilizan estos valores para estudiar la calidad de cada solución parcial, considerándola de una forma global, mediante las estimaciones de grafo. Si estas estimaciones no son lo

suficientemente fiables, el particionamiento dará como resultado un sistema aparentemente válido, pero que no coincidirá con sus verdaderas características, por lo que puede ser necesario repetir parte del proceso.

Toda esta etapa se ve dirigida por la denominada *función objetivo.* Esta función, valora todos los factores comentados anteriormente, y representa el inverso de la calidad del sistema que se está codiseñando. De esta manera lo que se pretende es minimizarla, con el fin de obtener la solución óptima. Es evidente que la elección de la función objetivo resulta crucial, ya que de no realizarse con cuidado se puede efectuar un particionamiento guiado de una forma no eficiente.

Existen numerosos algoritmos de particionamiento, los cuales han sido utilizados en muy distintos campos de aplicación [AlKa95]. Una labor importante es adaptarlos de una forma adecuada para que tengan en cuenta las características básicas del proceso de Codiseño, y por tanto que sean capaces de ofrecer los mejores resultados posibles.

1.2.4. Revisión de los resultados y generación del sistema.

Una vez acabado el proceso de particionamiento, y como se comentó anteriormente, es posible que el sistema obtenido no cumpla las condiciones impuestas en la especificación inicial. Esto puede deberse a un funcionamiento poco preciso de cualquiera de los módulos anteriores, y por lo tanto, sería necesario repetir el proceso total o parcialmente.

Para llevar a cabo esta comprobación es preciso realizar una *simulación del sistema* [AFSS98] [Shob98] y estudiar sus características sobre un conjunto de datos de entrada estándar. Evidentemente no es rentable proceder a una implementación final del sistema debido a la gran cantidad de tiempo que esta labor consume [HiBo97] [SuHa98]. Para realizar esta simulación es conveniente llevar a cabo técnicas relacionadas con el prototipado rápido [HaRo97] [PLCS97] [VaCJ97] [VeLi97], ya que es

importante que esta fase no suponga un gran incremento dentro del tiempo final del diseño.

Cuando el sistema generado cumpla los requisitos impuestos, la última etapa que resta por hacer es generarlo [CHMP98] [Liem97]. En este punto se dispondrá de dos particiones disjuntas: la partición software que dará lugar a un programa especificado en el lenguaje de alto nivel escogido, y que se ejecutará en el procesador del sistema, y la partición hardware, cuya estructura habrá que expresar en un lenguaje de descripción válido y que posteriormente dará lugar al ASIC [VeVV99].

Junto a estos dos componentes, hay que generar también un sistema de comunicaciones [LMTJ98] [NiMa98] [OnOJ98], formados por los interfaces hardware y software con el bus para el paso de información, y el control necesario para su uso.

Una vez finalizada esta tarea, se está en disposición de realizar una implementación física del sistema para verificar su buen funcionamiento en tiempo real.

1.3 Trabajos previos.

Aunque existen muchos estudios sobre el proceso de Codiseño, la mayoría de las aproximaciones carecen de ciertas características básicas que permitan llevar a cabo la tarea de una forma realista. Esto es muestra de lo alejado que se encuentra todavía el campo de investigación en este ámbito del entorno industrial encargado de la realización de diseños reales [HuJe98].

Así, consideraciones como la existencia de múltiples implementaciones hardware posibles para cada nodo, el paralelismo entre tareas no dependientes y el reuso de unidades funcionales, son muchas veces ignoradas por motivo de una mal entendida simplicidad.

De esta manera, y en lo que al proceso de estimación se refiere, muchos sistemas generados recurren a metodologías triviales, como la suma directa del coste de los distintos módulos [ErHB93], o el reuso exclusivo de instancias provenientes de una misma entidad hardware [NiMa96]. No hay que decir que evidentemente estos métodos consumen muy poco tiempo, pero también es cierto que pierden la precisión y realismo necesarios en este tipo de trabajos.

Por otra parte, y hablando del proceso de particionamiento, la mayor parte de los sistemas existentes se basan en algoritmos estándar que fueron desarrollados bajo condiciones y requerimientos distintos a los que poseen los problemas típicos que el Codiseño intenta solucionar. Es necesario destacar la importancia de que este proceso se lleve a cabo de una manera automática, para que diseños con un alto grado de complejidad puedan alcanzarse en el tiempo exigido por las necesidades de mercado.

De esta manera, sería preciso introducir una serie de innovaciones en dicha tarea que permitiesen abordar las particularidades inherentes de estos problemas de una forma eficaz.

A continuación se revisan los sistemas relacionados aparecidos en la literatura que se consideran más importantes, comentando el método que siguen a la hora de realizar el proceso de estimación y particionamiento, y poniendo en perspectiva aquellos detalles que deberían ser reconsiderados para obtener una solución aceptable y realista. Existen muchos más, pero la necesidad de centrar el problema hace que únicamente se detallen los siguientes.

1.3.1. Ptolemy.

El sistema *Ptolemy* [Kala95] [KaLe93] [KaLe94] [KaSu97] fue desarrollado en la Universidad de Berkeley por Kalavade y Lee. Su objetivo principal es el tratamiento de la Síntesis a nivel de sistema para procesado

de señal. De entre los trabajos existentes en la literatura actual, quizás es este el que se aleja más de los métodos y algoritmos estándar, para adentrarse en el estudio de nuevas heurísticas originales especialmente creadas para este problema.

Como en todos los casos en los que hay presente un mecanismo heurístico, la técnica propuesta es muy subjetiva, lo cual no desmerece su calidad, sino que simplemente aporta una solución determinada a un problema concreto.

El enfoque original del proceso de Codiseño aquí tratado reside en decidir qué módulos, de entre una especificación inicial en software, hay que migrar a hardware para cumplir una restricción temporal impuesta, así como su planificación temporal, de tal manera que el área total del sistema sea mínima. La idea fundamental del trabajo consiste en la utilización de una *función objetivo variable* (noción también presente en [DiJh97]), *en lugar de la fija usada en la mayoría de los sistemas*, que vaya cambiando dependiendo del estado general del proceso. De esta manera se pueden caracterizar de una forma más adecuada las necesidades de ahorro de tiempo o área del problema. Así, si el sistema se encuentra muy lejos de cumplir la restricción temporal se optará por reducir el tiempo, y en caso contrario se tratará de ahorrar área.

El algoritmo que lleva a cabo esta tarea se denomina GCLP (*Global-Criticality/Local-Phase*). Este nombre viene dado por los dos factores, *Criticalidad Global* (GC) y *Fase Local* (LP) que se encargan de decidir la mejor política de optimización de entre las dos comentadas anteriormente:

- La *Criticalidad Global* es una medida del estado general del sistema. Dada una solución parcial, indica qué porcentaje de los nodos que están todavía sin implementación sería necesario mover a hardware para cumplir la restricción temporal. Un valor de CG próximo a 1 sugiere que dicha restricción está todavía muy lejos de cumplirse, y por lo tanto

implica que la mayoría de los nodos deberían ir a hardware. Se tendrá exactamente el caso opuesto cuando GC sea cercano a 0. Para realizar el cálculo de este factor se van escogiendo los nodos sin asignación de una lista de prioridad P_f, y se van moviendo a hardware hasta que la restricción impuesta sea cumplida.

- La *Fase Local* refleja, para cada nodo, sus propias características internas, pudiendo de esta manera crear tres conjuntos independientes:
  a) *Extremos*. Un nodo es considerado un extremo respecto a una implementación *I* si necesita una gran cantidad de recursos típicos de esa implementación (tiempo en software y área en hardware) y muy pocos de la implementación opuesta. Se puede decir que los nodos de este tipo no despiertan ninguna duda en cuanto a qué partición serán asignados finalmente, ya que sus características internas fuerzan esta decisión.
  b) *Repulsores*. El concepto de repulsor se aplica a un par de nodos entre sí. Son aquellos con características muy similares en una implementación dada, pero donde uno de ellos tiene una necesidad mínima de recursos en la partición opuesta. En el caso de que hubiera que mover uno y sólo uno de los dos a dicha partición, sería este último el elegido, debido a las características intrínsecas comentadas de ahorro de recursos.
  c) *Normales*. Son aquellos nodos que no pertenecen a ninguna de las clases enumeradas anteriormente.

El sistema va eligiendo nodos secuencialmente, según una lista de prioridad, y se van calculando los factores LP correspondientes. Para cada uno de estos nodos, se trata de decidir la partición más adecuada, así como su planificación. Es evidente la necesidad de evaluar en cada paso una función objetivo, que indique la calidad del movimiento efectuado. Como se comentó anteriormente, la novedad de este sistema es su dualidad respecto

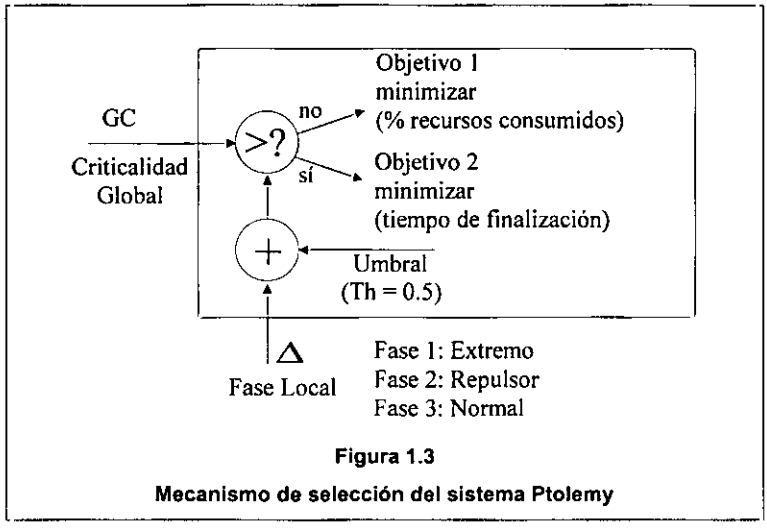


Figura 1.3

Mecanismo de selección del sistema Ptolemy

a esta función objetivo. A la hora de decidir cuál de ellas usar, se examina la siguiente expresión, relativa al nodo considerado en ese momento:

$$GC > LP + 0.5$$

Si esta relación es verdadera, se intentará minimizar el tiempo, ya que indicaría que la restricción temporal todavía está lejos de cumplirse. Recíprocamente, si la relación es falsa, se seguirá una política de reducción de área, ya que el tiempo no sería un problema en este caso (Véase Figura 1.3).

La principal carencia de este sistema reside en el proceso de cálculo del área. Aunque es preciso hallarla para realizar la evaluación de calidad del sistema, no se indica nada acerca del posible reuso de unidades hardware, por lo que parece que no se contempla esta posibilidad. Este hecho conlleva una estimación desmesurada del coste final, que probablemente no reflejará la realidad del sistema.

También se desprende cierto aire de subjetividad en las heurísticas que guían el proceso. El número de propiedades para hallar los repulsores, así como el umbral para considerar un nodo como extremo son ambiguos, lo que deja a la tarea en un estado de dependencia de estos parámetros.

Otro defecto que se podía encontrar en el sistema era el no considerar las posibles múltiples implementaciones hardware para un único nodo, algo altamente importante si se quiere dotar al proceso de un cierto realismo. Esto se solucionó con una mejora, que no sólo trata de abordar el problema inicialmente presentado (particionamiento y planificación), sino que además incorpora la asignación a cada nodo de su supuesta implementación óptima.

Para ello se propuso el algoritmo denominado IBS (*Implementation-Bin Selection*) [KaLe95] [KaLe97], que se ejecuta alternativamente con el GCLP. En este caso, cuando se elige un nodo para ser movido entre particiones, se calcula qué implementación, de entre un número finito previamente escogido, sería la más beneficiosa para mantener bajo el coste del sistema.

De esta manera, se calcula, para cada una de estas posibles implementaciones, qué porcentaje de nodos existentes tendrían que utilizar su implementación más costosa para cumplir las restricciones temporales. Con esto se obtiene una curva que refleja cómo afectará la decisión que se tome acerca de la implementación del nodo en cuestión en el resto del sistema.

Por lo tanto, valores altos en dicha gráfica implicarían un excesivo coste asociado al resto de nodos del sistema, y en consecuencia, dicha implementación no sería aconsejable para el nodo considerado.

Valores bajos de la gráfica indicarían el caso contrario. Así, el algoritmo selecciona, como supuesta implementación óptima para el nodo tratado, aquella con una pendiente mayor en la gráfica. Esto indicaría que la siguiente posible implementación tendría una carga de coste excesiva sobre el sistema, como se ha comentado en el párrafo anterior.

Esta solución es totalmente heurística, y aunque razonable, no garantiza que el proceso vaya a poder escapar de los mínimos locales. Además, se sigue sin afirmar nada acerca de cómo se realiza el cálculo del coste del sistema, y si se tiene en cuenta la compartición de hardware o no,

que al no ser mencionada es de suponer que no se tiene en cuenta dentro del proceso.

1.3.2. CASTLE.

CASTLE (Codesign *A*nd *S*ynthesis *TooL* *E*nvironment) [CaWi95] [PIGM98] [PIWi94] [PIWi95] [TVVV95] es un marco de trabajo desarrollado en el GMD de Bonn, Alemania, que engloba varias herramientas orientadas hacia el Codiseño y en el que, entre otras particularidades, se considera la división de las tareas software entre varias particiones.

Una de sus características básicas viene dada por una interacción constante entre los procesos que se llevan a cabo de forma automática y el diseñador que toma las decisiones relacionadas con su desarrollo.

En [ThSV94] se citan varios factores que definen el entorno, y que todos los sistemas deberían cumplir, pero se remarcan tres de ellos, que a juicio de los diseñadores son exclusivos de *CASTLE*:

1. Trabaja con una amplia variedad de arquitecturas hardware. Esto da la posibilidad de experimentar con distintas configuraciones para observar los resultados y decantarse por la más apropiada. De esta manera, no sólo se puede variar el tipo del procesador, sino cambiar su número y modelo de comunicaciones.
2. Muestra al diseñador datos relevantes sobre la implementación parcial. Esto incluye las estimaciones de tiempo y coste y la sobrecarga por comunicaciones entre módulos. Con esta información se tienen más elementos de juicio a la hora de tomar decisiones.
3. Admite el mantenimiento de la implementación. Si se producen cambios en la especificación del sistema, obviamente influirán en el resultado final. Es conveniente que, en este caso, no se vuelva a implementar todo el producto desde el principio, sino que sólo se vean tratadas las partes implicadas en la modificación.

Estas tres características que ofrece el entorno, si bien son positivas, realmente no representan nada especial dentro del conjunto de sistemas existentes. El hecho de poder seleccionar la arquitectura es algo que en todas la aproximaciones es necesario realizar. Aunque aquí se ofrezca un conjunto destacado, es la experiencia del propio diseñador la que marcará la tendencia más apropiada en cada caso.

El disponer de los datos intermedios generados por el sistema es una concepción trivial de realimentación de datos. Es evidente que en todo sistema la información ha de estar disponible, para que se pueda controlar el proceso, y en su caso interaccionar con él cuando sea conveniente.

Por último, se menciona el concepto de modularidad, que al tratarse de un entorno integrado por herramientas provenientes de distintos ámbitos, es lógico que se posea.

El sistema, una vez comenzado el proceso de Codiseño, lleva a cabo una primera etapa consistente en la realización de un análisis del diseño. En él, se estudia el programa que sirve de especificación, y que puede venir dado en C++ o VHDL, y se separa en bloques básicos.

Después, se realiza un perfil de cada bloque, utilizando para ello cargas de datos típicas de las aplicaciones que se vayan a tratar. Con esto, se calcula el número de veces y la probabilidad de que estos bloques se ejecuten.

Además, se realiza un análisis estático en el que se obtienen características como las dependencias de datos, información transmitida entre funciones y la existencia de estructuras de datos complejas, como punteros o *arrays*.

Una vez acabado este análisis se procede a realizar la tarea de particionamiento, que se divide en varias fases:

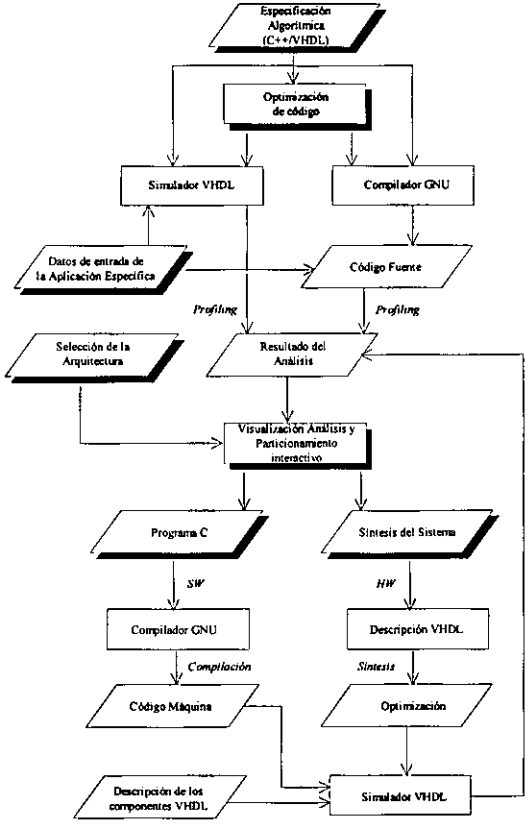


Figura 1.4

Esquema general del sistema CASTLE

1. Elección de la arquitectura objetivo. De entre una lista dada, el diseñador escoge aquella arquitectura que le parezca más interesante. Además, se ofrecen varias posibilidades en cuanto a mecanismos de comunicación y sincronización.
2. Reestructuración de los componentes. Se da libertad al diseñador para fundir o separar funciones que a su criterio deben implementarse conjuntamente, con lo que se modifica la jerarquía de la especificación.

3. Asignación de componentes. El diseñador asigna, una a una, todas las funciones a la partición software o a la partición hardware, según crea más conveniente en base a la información suministrada por el sistema.

La cuestión fundamental es que este proceso se desarrolla de forma manual, es decir, es el diseñador, basándose en conocimientos particulares, el que dirige al sistema en la migración de componentes entre las particiones.

Evidentemente, si la complejidad del problema es grande, este método podría no ser el más adecuado, ya que el espacio de búsqueda crece considerablemente, y por lo tanto también lo hace la dificultad para encontrar la solución óptima.

De esta manera, si se observa el último punto del proceso de particionamiento, se llega a la conclusión de que una de las tareas más complejas dentro de toda metodología de diseño, la asignación de componentes, se realiza de forma manual, lo que claramente restringe las posibilidades de una exhaustiva búsqueda dentro del espacio de las soluciones.

Además, esto redunda en una disminución de la automatización, que como se explicó es necesaria para reducir el tiempo de diseño.

En la Figura 1.4 se muestra un esquema general del sistema *CASTLE*. Los símbolos sombreados corresponden a las herramientas específicamente diseñadas para el entorno, mientras que el resto representan otro tipo de herramientas necesarias para su buen funcionamiento.

1.3.3. COSMOS.

El sistema *COSMOS* [IsJe95] [MaDJ97] [ObIJ93] [VIJK94] es un entorno para Codiseño desarrollado en el Instituto Politécnico Nacional de Grenoble. Mediante una serie de primitivas previamente definidas, se interactúa con el proceso, guiando la migración de bloques entre el

hardware y el software. Realmente, este tipo de particionamiento se puede considerar manual, porque las decisiones del diseñador pesan tanto en la obtención de la solución final que se hacen imprescindibles en esta tarea.

El entorno está basado en *Solar* [IsJe95], un modelo de representación interna para sistemas, el cual permite varios niveles de descripción. Su estructura básica es una máquina de estados finitos con posibilidad de representar jerarquía y paralelismo, y posee claras reminiscencias de las metodologías de interconexión de procesos en el lenguaje VHDL.

El proceso de Codiseño consta de varias etapas. En la primera de ellas, se parte de una descripción del sistema en el lenguaje SDL, que viene representado por una serie de procesos intercomunicados. A continuación se transforma a la estructura de *Solar*.

En este punto, la representación estará compuesta por una serie de *unidades de diseño* (DU), comunicadas mediante *unidades de canal* (CU). Estas unidades pueden estar modeladas mediante los denominados *operadores de tabla de estados* (ST).

Las unidades de diseño corresponden a unidades de proceso de operaciones, que más tarde darán lugar a las correspondientes unidades funcionales. Las unidades de canal representan distintos modelos de comunicación entre los procesos existentes. Por último, las tablas de estado tienen relación con las distintas características que expresan el comportamiento de los sistemas secuenciales mediante la generación de máquinas de estado finitas. La traducción desde SDL a *Solar* es muy sencilla, debido a la similitud de sus características.

Después, se efectúa la fase fundamental, que es la de particionamiento. Esto se lleva a cabo mediante una herramienta especializada en dicha tarea, conocida como *PARTIF* [IsOJ94]. Hay que destacar que esta no se comporta de una forma automática, sino que ha de

ser guiada por el diseñador mediante el uso de cuatro primitivas definidas para ello:

- *MOVE:* Permite reordenar la jerarquía de la estructura interna, moviendo procesos de un punto a otro.
- *MERGE*: Realiza la fusión de dos procesos en uno solo, con lo que se facilita la compartición de recursos.
- *SPLIT:* Transforma una máquina secuencial en un conjunto de máquinas paralelas, con lo que se acelera la ejecución de las tareas.
- *CUT:* Transforma un conjunto de procesos paralelos en un conjunto de procesos intercomunicados a través de canales.

La salida de esta fase es un nuevo modelo formado por las denominadas *unidades de proceso abstractas* (AP), que previamente han debido ser asignadas a hardware, software o microcódigo adaptado para un procesador estándar, así como por unidades de comunicación.

El siguiente paso que se realiza es la síntesis de comunicaciones. De esta manera, se sustituyen los canales del sistema por señales reales, junto con su control asociado. Para realizar esta tarea se utilizan las librerías internas de *Solar*, compuestas por módulos reales, las cuales reciben posteriormente una implementación física.

Por último, se lleva a cabo la generación de la arquitectura. Se parte de dos subsistemas hardware/software intercomunicados, sobre los que se realizan dos tareas:

- Un prototipado virtual, que consiste en la generación de código para cada uno de los subsistemas. Se utiliza VHDL para la parte hardware y C para la software.
- Un mapeado de la arquitectura, mediante generadores estándar de software y herramientas de síntesis para hardware.

Las críticas que se pueden verter sobre este sistema, vienen encaminadas en la doble dirección del particionamiento y las estimaciones.

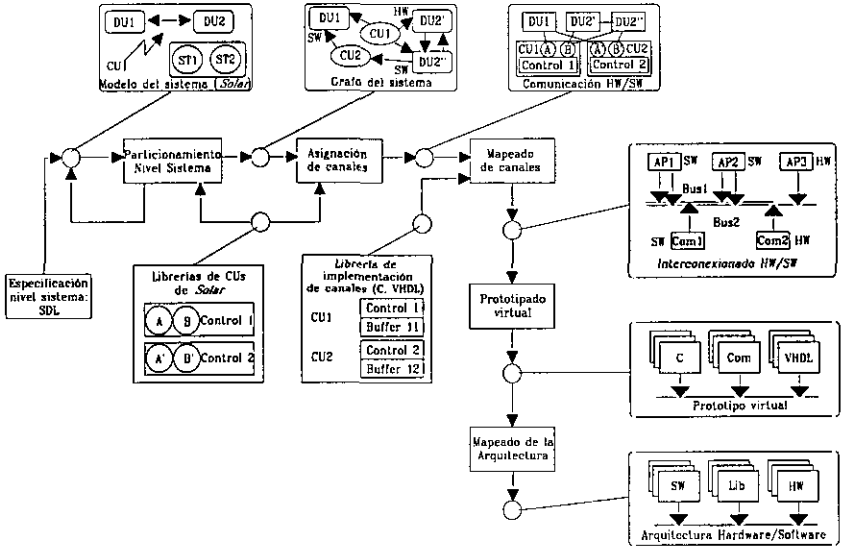


Figura 1.5

Esquema general del sistema COSMOS

Acerca de este último proceso, es posible decir que el sistema de asignación de hardware a las distintas funcionalidades contenidas en la descripción inicial se reduce a simples mapeados de dichas operaciones sobre módulos estándar contenidos en una biblioteca.

El reuso que se puede realizar en estas circunstancias es muy simple y trivial: un módulo se puede compartir entre dos procesos si estos dos procesos son instancias idénticas de una misma entidad, o dicho con otras palabras, si son absolutamente iguales. Sin embargo, casos de módulos altamente parecidos, en los que su única diferencia radica en la presencia de un conjunto mínimo de operaciones, no pueden ser reusados en absoluto, con lo que esta técnica desvirtúa claramente la realidad de este tipo de procesos.

Por otra parte, el particionamiento, que se plantea como un proceso interactivo, esencialmente se desarrolla de una forma manual. Esto es

debido a que no se propone ningún tipo, ya no de proceso automático, sino de heurística que permita la aplicación razonable de las primitivas expuestas anteriormente.

De igual manera, no se proporciona una función de coste que evalúe el resultado de lo estados parciales, por lo que el diseñador se ve en la circunstancia de realizar una búsqueda ciega en el espacio de soluciones.

En la Figura 1.5 se ofrece un esquema general del sistema COSMOS.

1.3.4. COSYMA.

El sistema *COSYMA* [ErHB93] [HEHB94] [HoEr93] [YEBH93] (*COSY*nthesys of e*M*bedded *A*rchitectures), desarrollado en la Universidad Técnica de Braunschweig, ofrece un entorno en el cual poder realizar el proceso de Codiseño partiendo de una orientación inicial totalmente en software.

COSYMA fue uno de los sistemas pioneros en este aspecto, aunque carece de ciertas características básicas ampliamente consideradas hoy en día, como la compartición hardware y la ejecución paralela. Aún con esto, sigue siendo uno de los modelos de sistema con granularidad fina más referenciados.

Inicialmente, se parte de una descripción del sistema que viene dada en una ampliación de C, denominada C^x, en la que se pueden especificar los conceptos relacionados con las restricciones temporales. Esta decisión se debe a que los lenguajes estándar de descripción software carecen de ciertos mecanismos básicos para tratar la representación hardware, como la sincronización o la temporización.

Después de esta etapa, la especificación inicial se compila a una estructura interna que recibe el nombre de *grafo de sintaxis extendido* (ESG). Esta estructura está formada por dos grafos dirigidos acíclicos, coexistentes sobre los mismos nodos. El primero representa el orden de las

operaciones según fueron definidas en la especificación de partida. El segundo indica las dependencias de datos entre dichas operaciones. De esta manera, se obtiene una representación interna del problema en la que está contenida toda la información necesaria para llevar a cabo el proceso.

Posteriormente, se lleva a cabo un análisis rápido de la aplicación, por medio de una simulación, para obtener la frecuencia de iteración de cada bloque [YEBH93], y después se realiza el proceso de estimación para guiar el particionamiento. En este punto es donde se pueden aducir las mayores objeciones al sistema, debido a la falta de rigurosidad y realismo. Si bien, como se ha comentado anteriormente, estas ideas fueron pioneras en su género, esto no es óbice para la utilización de un método de estimación tan rudimentario.

Este proceso se calcula de una forma iterativa, es decir, basándose en el resultado de la etapa anterior. Así, al mover un bloque de software a hardware, el incremento de tiempo producido se representa por:

$$\Delta(b) = a(T_c, T_s)*[t_{neff}(B)+t_{com}(B)-t_{HW\text{-}SW}(B)-t_{SW}(B)]*It(B)$$

$a(T_c, T_s)$ es un factor dependiente de la restricción temporal que mide cuán grande es la violación de dicha restricción. Con esto, se pretende penalizar en exceso soluciones que no son válidas para que de esta manera sean automáticamente rechazadas por el particionador.

$t_{neff}(B)$ y $t_{SW}(B)$ son el tiempo de ejecución del bloque en hardware y software respectivamente.

$t_{com}(B)$ es el tiempo de comunicación entre las dos particiones, que se verá afectado tras el movimiento del bloque a hardware.

$It(B)$ es el número de veces que se repite el bloque, estimado mediante el análisis por simulación comentado anteriormente.

$t_{HW\text{-}SW}(B)$ es el intervalo de tiempo en el que ambas particiones solapan su ejecución, o dicho de otra manera, el tiempo en el que se produce una ejecución paralela del sistema.

El último factor comentado concerniente a la posibilidad de paralelismo se presenta a nivel teórico, pero es ignorado en la evaluación práctica. Esto es sinónimo de afirmar que la ejecución paralela de distintas partes del sistema no es considerada, con la consiguiente sobrestimación del tiempo final.

Además, la estimación de coste realizada es extremadamente simple. Se basa en un proceso aditivo trivial en el que el coste de los distintos módulos se suma para hallar el coste total del sistema. Obviamente, esto es rechazable por cualquier aproximación real, donde se considere la posibilidad de reuso hardware.

Una vez hallada la supuesta mejora producida por un movimiento, se sigue realizando el proceso de particionamiento de una forma alternativa mediante el algoritmo de *Simulated Annealing* [KiGV83]. COSYMA tiene la característica de que la granularidad, o tamaño de los bloques básicos que se mueven en este proceso, es muy fina. Realmente se baja hasta el nivel de operación, con lo que el número de componentes que se obtiene para una especificación inicial es muy grande.

En la mayor parte de los problemas es discutible la elección de una granularidad tan baja, ya que cualquier caso real es inabordable mediante esta política debido al gran número de nodos que se obtendrían. Además, se carecería de una visión global al intentar implementar operaciones básicas por separado, con la consiguiente pérdida de la semántica inicial.

Para el caso de contar con un alto número de nodos, como ocurre en COSYMA, el algoritmo de *Simulated Annealing* da buenos resultados, ya que al basarse en un proceso de aleatoriedad, el hecho de tener una amplia muestra de base favorece la obtención de una solución adecuada. Sin

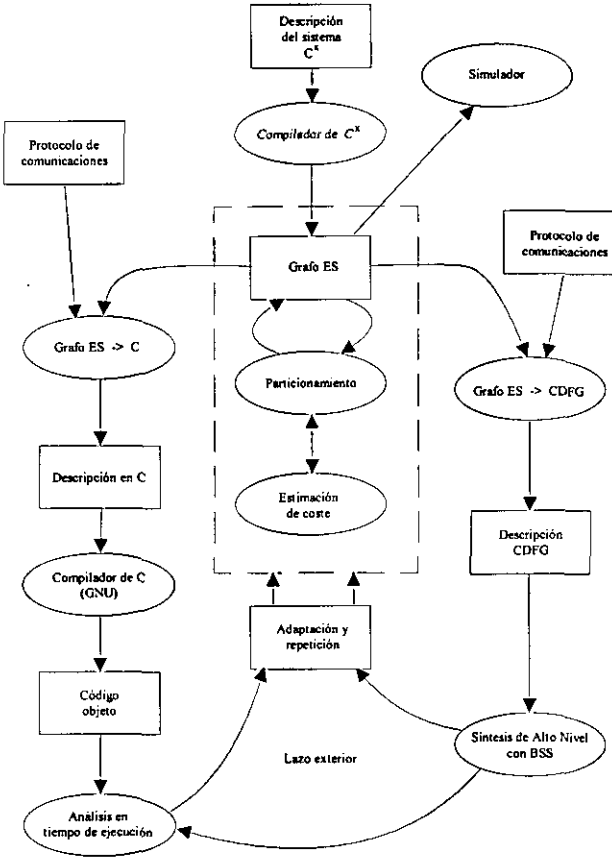


Figura 1.6

Esquema general del sistema COSYMA

embargo, el tiempo de ejecución que necesita el algoritmo para llegar a dicha solución es muy grande, debido al elevado número de pasos que hay que realizar para que converja.

Además, la complejidad intrínseca de este algoritmo hace que no se garanticen unos buenos resultados, a no ser que se adapte especialmente para cada aplicación.

Una vez realizado este proceso se evalúa la solución obtenida, y en el caso de que no sea la esperada, el sistema se realimenta mediante el denominado *lazo exterior* [HeHE94]. En él, se estudian los errores entre la estimación realizada y los valores finales verdaderos. Mediante esta comparación, se pretende hallar unos nuevos valores más ajustados en la estimación inicial para obtener futuros resultados más acordes a la realidad.

Finalmente, una vez acabada la etapa de particionamiento de una manera satisfactoria, se procede a la generación del sistema de la siguiente forma:

a) Un programa en C para la parte software, que corresponde a un subconjunto de la especificación inicial, y que servirá de entrada a un compilador estándar de GNU.

b) Un circuito hardware, que puede ser generado de dos posibles formas:

- Mediante el sistema de Síntesis de Alto Nivel *OLYMPUS* [DeKM90], el cual acepta como entrada el lenguaje Hardware-C.
- Mediante el coprocesador específico BSS (*Braunschweig Synthesis System*) [HEHB94]. Esta opción tiene la ventaja de reducir considerablemente el tiempo de diseño, ya que se utiliza un módulo específicamente implementado para ello.

Una vez generado el sistema se procede a su verificación mediante un análisis en tiempo real. De no cumplir las exigencias impuestas, se examinan los pasos anteriores de diseño para corregir cualquier desviación en el proceso.

En la Figura 1.6 se ofrece un esquema general del sistema COSYMA.

1.3.5. SpecSyn.

El sistema *SpecSyn* [GaVN94] [GGVN93] [NaGa94] [NaVG92] es un entorno desarrollado en la Universidad de California en Irvine, y que engloba diversas herramientas que configuran una metodología de diseño.

Tiene la característica de que su entrada viene dada en forma de especificación ejecutable, con el fin de conseguir las siguientes ventajas:

- Facilidad en la simulación para verificar la corrección del sistema durante las primeras etapas del diseño.
- Introducción directa de la especificación en herramientas de síntesis, con el consiguiente ahorro de tiempo.
- Utilización de la propia especificación como documentación del sistema, proporcionando una descripción precisa de su funcionamiento.

El modelo que soporta esta especificación se denomina *Máquinas de estado de programas* (PSM). Su particularidad se halla en la combinación de máquinas de estado finitas jerarquizadas con paradigmas de los lenguajes de programación.

De esta manera, un sistema viene especificado como una jerarquía de *estados de programas*, los cuales pueden incluir declaraciones estándar, como variables, tipos, subrutinas y otras similares.

Las tres clases de objetos que comprenden las especificaciones son: *variables, comportamientos* y *canales*. Las primeras tienen la función de almacenar datos, los segundos de transformar dichos datos, y los terceros, de comunicar varios comportamientos entre sí.

Las etapas fundamentales que se realizan en el proceso son:

1. *Asignación:* Añade componentes al sistema, como memorias, registros, procesadores estándar o ASICs.
2. *Particionamiento:* Mapea cada clase de objetos funcionales a los componentes asignados anteriormente. Las variables se mapean a posiciones de memoria, los comportamientos a procesadores estándar o específicos, y los canales a buses.
3. *Refinamiento:* Añade nuevos comportamientos para mantener el correcto funcionamiento del sistema, que se puede ver modificado después de realizar las dos etapas previas.

La crítica que se puede realizar a este sistema está en la línea de los anteriores, donde el proceso de asignación de hardware requería de una gran intervención por parte del diseñador. De esta manera, si es necesario añadir una serie de elementos estándar al diseño, para que sustente la realización de las funcionalidades, se está incurriendo en dos factores negativos: por una parte, se aumenta considerablemente el tiempo de diseño, lo que va en contra de los objetivos especificados; por otra parte, se está perdiendo una gran versatilidad del hardware, ya que las posibilidades de generación de circuitos óptimos se ven restringidas por la adición de estos elementos estándar.

A la hora de realizar el proceso de particionamiento, el sistema utiliza un algoritmo denominado *BCS* [VaGG93] [VaGG94], basado en el proceso de búsqueda binaria. El objetivo que se plantea es, dado un rango de costes hardware, encontrar aquel que sea mínimo y que respete una serie de restricciones de tiempo impuestas.

Para ello, se parte de dos soluciones que a primera vista no son satisfactorias, debido a que no cumplen las restricciones temporales, o bien se intuye que su coste es demasiado alto. Si ambas soluciones están situadas en puntos extremos del espacio de diseño, es de suponer que la mejor solución se encontrará entre estas dos propuestas. De esta manera, se escoge el punto medio de coste entre las dos, y se intenta hallar una solución válida asociada mediante el algoritmo de *Simulated Annealing*. Si se consigue, se repite el proceso, pero ahora con el subintervalo inferior al actual punto tratado. En caso contrario, se optaría por seguir buscando en el subintervalo superior. El algoritmo concluye al encontrar una solución que se considere óptima.

Esta metodología es la que se conoce como búsqueda binaria. Sin embargo, el algoritmo plantea una gran duda acerca de su fiabilidad: los costes que se pueden alcanzar en un sistema no están distanciados

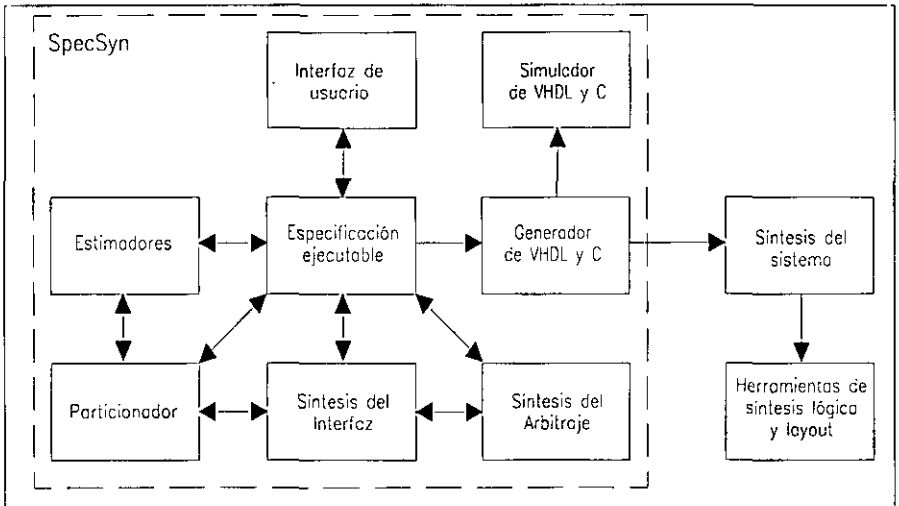


Figura 1.7

Esquema general del sistema SpecSyn

uniformemente, sino que son discretos, y dependen de factores diversos como las características de los módulos que se utilicen.

Al no existir una separación predeterminada en el rango de costes, no se puede hablar de un incremento fijo, necesario dentro del algoritmo de búsqueda, y tampoco se menciona nada acerca de cómo solventar este problema.

Además, no se indica qué criterio se utiliza para seleccionar las soluciones que se evalúan por el algoritmo, ni cómo comprobar la existencia de una de ellas que cumpla las restricciones impuestas.

En la Figura 1.7 se muestra un esquema general del sistema *SpecSyn*.

1.4 Estructura del trabajo desarrollado.

Una vez presentado a lo largo de este capítulo la introducción, composición y trabajos previos de la metodología de Codiseño, es necesario explicitar la estructura de la investigación desarrollada en este ámbito.

Inicialmente, y debido a la gran variedad de aproximaciones hacia el Codiseño, como ha debido quedar patente tras la exposición de este capítulo, centraré el entorno de trabajo, junto con las consideraciones adoptadas a la hora de realizar esta investigación (§2). Después, ofreceré una crítica detallada del punto de vista con el que la mayoría de las metodologías abordan los problemas hoy en día. Esto dará lugar a una nueva concepción del Codiseño, que es la que ha originado la idea del presente trabajo (§3).

Posteriormente, explicaré el fundamento teórico que caracteriza la nueva aproximación, así como el efecto que tiene en el importante proceso de estimación de parámetros (§4, §5). Es obvio que toda esta innovación repercute de forma directa en el núcleo de la metodología de Codiseño, el particionamiento, por lo que detallaré las variaciones introducidas, así como una novedad en la forma de entender los algoritmos asociados (§6).

Finalmente, expondré el entorno experimental desarrollado para verificar prácticamente estas teorías, con los resultados obtenidos en este proceso (§7), así como las conclusiones finales y el trabajo pendiente generado por toda la investigación (§8).

Delimitación del problema de Codiseño: Consideraciones previas.

2

Tras la introducción al Codiseño presentada en el capítulo anterior, se debería haber llegado a la conclusión de la gran complejidad asociada a esta metodología. No solamente eso, sino que además la enorme variedad de aproximaciones de que se dispone, dependiendo de las características propias de cada problema, hace difícil la estandarización de los procesos y sus técnicas. Así, de la misma forma que es posible utilizar el Codiseño para conseguir sistemas que cumplan una cierta restricción temporal, es igualmente válido aplicarlo a problemas que intenten minimizar el coste de un determinado circuito.

Esto añade uno de los grandes problemas existentes en la comunidad científica dedicada al estudio del Codiseño, y en un sentido más general, a la Síntesis de Alto Nivel: las muy distintas consideraciones y la variada semántica utilizada bajo los mismos conceptos. En este capítulo trataré de definir las bases del entorno de trabajo presentado para llevar a cabo esta investigación, o con otras palabras, *qué se ha entendido por Codiseño*. Así, se pretende que el lector llegue a los capítulos posteriores, donde se explica la teoría desarrollada, con las ideas centradas en el tema de trabajo, descartando otras aproximaciones igualmente válidas a esta metodología.

2.1 La variedad de concepciones acerca del Codiseño.

Una de las primeras dificultades al introducirse en el mundo del Codiseño reside en la multiplicidad de aspectos y consideraciones distintas que es posible encontrar en la literatura. Estas diferencias, normales por otra parte, se hacen especialmente patentes en este campo, lo cual se puede deber a varias razones.

Por un lado, el Codiseño es una disciplina bastante reciente. Así, hace sólo cinco o seis años, el hecho de poder abordar la síntesis de sistemas complejos de una manera automática era un hecho casi impensable. Esto ha producido una carencia de estandarización, que ha conllevado una evolución de esta metodología hacia distintas direcciones, según el entorno en el que se esté desarrollando el proceso.

Un ejemplo de esta falta de acuerdo es la no existencia de un lenguaje de especificación unánimemente reconocido [Nebe99]. La forma en la que inicialmente se modele el sistema resulta determinante en etapas posteriores [ViVe98], como la elección de la granularidad, y hace que se tienda hacia soluciones radicalmente distintas.

Sin embargo, sigue existiendo controversia acerca de si lo más apropiado es utilizar un lenguaje de especificación software, uno de especificación hardware, o crear uno nuevo con características propias de descripción del sistema. Todas las aproximaciones son razonables pero hasta ahora ninguna ha demostrado ser lo suficientemente convincente como para convertirla en un estándar. Así, se han probado a utilizar lenguajes híbridos como el Hardware-C, que combinan la facilidad descriptiva de los procedimientos software con características esenciales físicas, como la temporización y la sincronización. Otras aproximaciones se

decantan por propuestas de descripción formales, como LOTOS [LCLS96], o por lenguajes de reciente creación, como Java [HeOl97].

No obstante, todavía es normal el hecho de encontrar que prácticamente la totalidad de los sistemas existentes proponen su propio método de especificación con particularidades únicas, dependiendo de lo que se esté tratando de expresar. Esto debe dar una idea de la gran heterogeneidad del Codiseño, aun partiendo desde sus etapas más tempranas.

Sin embargo, no es solamente la reciente aparición de esta metodología la que conlleva este tipo de situaciones, sino también el momento en el que ha surgido. Así, aún existen distintos aspectos de la Síntesis de Alto Nivel para los que todavía no se ha propuesto una solución realista, o sobre los que es posible realizar un conjunto de optimizaciones.

En estas circunstancias, cuando existían puntos difusos en las bases de la síntesis automática, se empezaron a tratar problemas con un grado de abstracción superior, y por lo tanto con la necesidad de utilizar sobre ellos metodologías más complejas y elaboradas. Pero para desarrollar estas nuevas técnicas, se han debido realizar en muchos casos suposiciones acerca de aspectos más básicos, todavía no resueltos, para poder progresar hacia problemas más complejos.

Son precisamente esas suposiciones acerca de características aún oscuras, las que han producido multitud de aproximaciones distintas, cada una de ellas simplificando aquellos puntos más convenientes para el desarrollo de la propia teoría. No se está criticando el hecho de que cada cual trate de amoldar las especificaciones y el entorno de trabajo lo más posible para favorecer una aproximación propia. Realmente, la ciencia debe avanzar basándose en el principio de simplicidad progresiva.

Asimismo, es favorable el comenzar a desarrollar nuevas metodologías de trabajo, como el Codiseño, antes de fijar todos y cada uno de los puntos

de la Síntesis de Alto Nivel. No en vano, siempre debe existir un cierto germen teórico inicial que vaya por delante del grueso de aplicaciones prácticas que se desarrollen, ya que esta exploración más allá de los límites del conocimiento que se produce en cada instante es el verdadero motor que apoya las sucesivas innovaciones. No obstante, todos estos puntos, sin duda positivos, tienen un efecto colateral de dispersión de las teorías, que hace difícil estudiar el Codiseño como una técnica unificada.

Otro de los factores que influyen en la variedad de esta metodología es su propia estructura en sí. En un principio, puede parecer que el paso desde un diseño automático de circuitos hardware a una síntesis de sistemas, es sencillamente la extensión de unos ciertos problemas a otros de una mayor complejidad. Sin embargo, este paso no conlleva sólo un cambio en el tamaño de los objetos que se tratan, sino que existen profundas diferencias conceptuales entre ambas aproximaciones.

Así, una de las grandes dificultades que encierra un sistema es su propia abstracción. De hecho, un sistema puede ser casi cualquier entidad del mundo real, y es esa falta de precisión y definición la que luego se va transmitiendo a lo largo de todo el proceso de Codiseño.

No solamente la propia ambigüedad del sistema hace difícil la regularización de la estructura, sino que las circunstancias que lo rodean ayudan a descontrolar aún más la claridad del proceso. De esta manera, las situaciones sobre las que es posible aplicar el proceso de Codiseño pueden ser muy dispares.

En consecuencia, y como ya se comentó en el capítulo introductorio, la versatilidad de disponer de dos subsistemas diferentes y complementarios, como son el hardware y el software, hace de esta metodología una técnica muy potente para abordar problemas tan distintos como el abaratamiento de un circuito o la mejora de rendimiento de un código que se ejecuta sobre un procesador estándar.

Además de esto, el hecho de que todas las entidades pertenecientes al campo de la Informática se cataloguen como hardware o software, hace que casi cualquier problema de síntesis se pueda abordar mediante el Codiseño. De esta manera, se ha propuesto en este ámbito la generación automática de ASIP's [BISH94] [HuDe94a] [HuDe94b] [SAHN94], en la que la decisión acerca de la arquitectura viene dada por la elección de la parte hardware, y la especificación del repertorio de instrucciones, por la parte software.

Una razón más para comprender la complejidad de este proceso, es que el rápido crecimiento del Codiseño ha hecho que se dispare el número de metodologías presentadas de una manera espectacular, lo que ha llevado a la dificultad para adquirir los puntos positivos y ventajas de los trabajos ajenos. Esto ha conllevado, en un principio, el crecimiento de numerosas ideas en paralelo, generando una gran diversidad de aproximaciones acerca de los mismos conceptos.

Con todo lo comentado hasta aquí, parece clara la necesidad de centrar el presente trabajo de Codiseño, a fin de evitar posibles confusiones con otras tendencias de la misma metodología. Los siguientes apartados están dedicados a explicitar cuáles han sido las consideraciones que han rodeado al proceso de investigación que aquí se ofrece, así como las simplificaciones, siempre necesarias, que se han tenido en cuenta.

2.2 Consideraciones del entorno de trabajo y simplificaciones realizadas.

Entre las consideraciones fundamentales que se deben asumir acerca del trabajo realizado se encuentra la *delimitación del campo de aplicación.* Es evidente que dependiendo de los requerimientos que se deban imponer al sistema codiseñado, las técnicas de trabajo harán más hincapié en la optimización de unos parámetros frente a otros.

Otra noción muy importante reside en la *determinación de la arquitectura objetivo* que se desee utilizar. Esta decisión influirá en el equilibrio de dos características básicas: la versatilidad y eficiencia del sistema frente a su coste y complejidad. Evidentemente, cualquiera que sea la opción final ha de constar de dos subsistemas hardware y software, pero su estructura y relación pueden variar dependiendo del problema sobre el que se vaya a trabajar.

Un último punto trascendente es la consideración del objetivo final de este trabajo, que no es otro que la propuesta de nuevas metodologías de estimación hardware aplicadas al proceso de particionamiento, todo ello comprendido dentro del marco general del Codiseño. Para favorecer este objetivo, se han sustituido los mecanismos que tratan *la parte software del sistema mediante un conjunto de simplificaciones adecuadas.* Este hecho no viene dado por la falta de importancia o interés en esta parte, sino por la necesidad de centrar y acotar un trabajo, que de otra manera hubiese sido demasiado largo y disperso.

Todas estas consideraciones y simplificaciones se verán detalladas en los siguientes apartados.

2.2.1. Delimitación del campo de aplicación.

Los campos de aplicación sobre los cuales puede trabajar el Codiseño son innumerables, especialmente si se tiene presente la muy amplia presencia de sistemas digitales en la vida cotidiana. Lejos de estancarse, esta presencia continúa creciendo, lo que incita a la ampliación y refinamiento de estas técnicas.

Sin embargo, y con ánimo de simplificar, estas áreas de trabajo podrían dividirse en dos grandes grupos: los sistemas con altas restricciones de tiempo, como los que se utilizan para realizar tareas en tiempo real, y los

sistemas que necesitan minimizar su coste de producción como objetivo principal.

Entre el conjunto de sistemas englobados en el primer grupo están todos aquellos cuyos resultados producidos no sólo tienen que ser precisos, sino que además dichos resultados no tienen valor fuera de un margen de tiempo definido. En consecuencia, muchas veces se prefiere disminuir su calidad (dentro de unos límites, por supuesto), a fin de *garantizar* el cumplimiento de dichas restricciones temporales. De esta manera, y dependiendo de lo estrictas que sean, los sistemas generados estarán asociados a tareas más o menos vitales.

Así, este tipo de sistemas se pueden encontrar en aparatos predictivos situados a bordo de vehículos, en los que las operaciones realizadas son críticas para el buen funcionamiento del mismo. Pero de la misma manera es posible hallarlos en lazos de control dedicados a regular plantas industriales, en los que el factor tiempo sigue siendo importante, pero no vital.

El segundo grupo de sistemas, formado por aquellos que requieren un coste mínimo, son los que están integrados, a veces de una manera minoritaria, en aparatos de uso doméstico cada vez más frecuentes. El hecho de querer minimizar el coste de producción tiene el sentido de intentar una mayor presencia en el mercado, al reducir de esta manera el precio final del producto.

El componente digital incorporado, por ejemplo, a cualquier electrodoméstico, tendría una mínima importancia respecto a la función en sí del aparato. En muchos casos, este circuito sería incluso accesorio, pero se decide incorporarlo por razones puramente comerciales. Así, se pretende que cumpla la función, usualmente no esencial, para la que se ha diseñado, pero sin aumentar su coste de fabricación, lo que aumentaría el precio final del producto.

Para el presente trabajo, se han elegido los sistemas digitales en tiempo real con altas restricciones temporales [BaSa97] [HuDa97] [MoDe97] [MRKD98], como los comprendidos en el primer grupo comentado. Las razones para esta elección han sido el *a priori* mayor interés de estos sistemas respecto al campo de investigación presente, y a su mayor dependencia hacia la metodología de Codiseño, ya que la existencia de restricciones de tiempo muy rígidas crea la necesidad de empleo de técnicas altamente específicas para generarlos.

De esta manera, los sistemas en tiempo real siempre se han venido especificando mediante el uso de un código software que se ejecuta sobre un procesador de propósito general. La mayoría de las veces, y debido a esas fuertes restricciones comentadas, el tiempo de ejecución del código sobre el procesador es inaceptable para el tipo de tarea que se está intentando realizar.

La solución tradicional para este grupo de problemas ha sido la inclusión de un nuevo procesador más potente, solución esta no muy eficaz, ya que las restricciones impuestas sobre la tarea no encuentran límite en cuanto a su exigencia, y por lo tanto, es fácil volver a encontrarse de nuevo en la misma situación poco tiempo después. Otras soluciones, como la simplificación de los algoritmos utilizados, tampoco se han mostrado excesivamente eficientes.

Utilizando una aproximación al Codiseño, lo que se pretende es proponer un método mucho más razonable, para lo cual se aboga por la introducción de un circuito hardware, comunicado con el procesador principal, que se encargue de ejecutar las funcionalidades que más tiempo consuman. Debido a la mayor rapidez del hardware respecto al software, se consigue de esta manera reducir el tiempo de ejecución del sistema, encontrando así una solución factible.

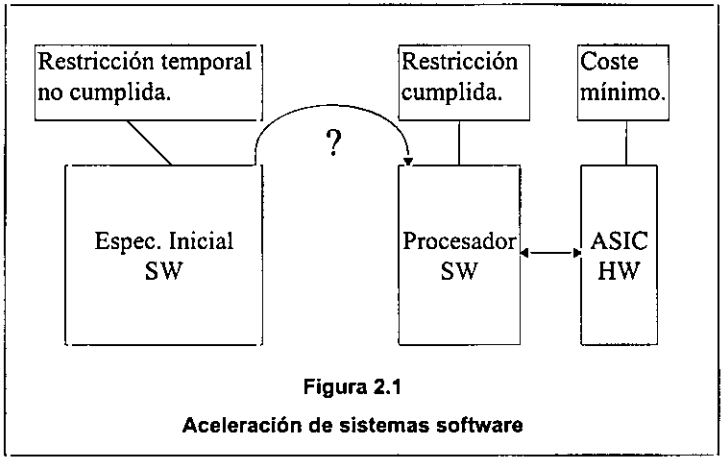


Figura 2.1

Aceleración de sistemas software

Sin embargo, el hecho de utilizar un único procesador con el código software necesario para realizar la tarea, tiene la ventaja de su muy bajo coste, característica esta que se debe seguir manteniendo. Para ello, el circuito hardware introducido ha de ser mínimo, y únicamente ejecutar aquellas porciones del código que realmente afecten a la eficiencia del sistema (Figura 2.1).

Por lo tanto, y para concretar, el objeto de entrada adoptado para desarrollar la metodología de Codiseño presentada es una especificación, pensada para ser ejecutada en un procesador estándar, así como una restricción temporal que se debe cumplir obligatoriamente.

El resultado del proceso de Codiseño será la relación de tareas que se siguen ejecutando en software, así como aquellas que deben ser implementadas en la parte hardware. Del mismo modo, se debe informar de cómo están relacionados estos dos grupos de tareas, junto con las características finales de tiempo y coste asociadas a cada una de las dos particiones.

En una metodología ideal, este resultado vendría acompañado tanto del código software final, subconjunto de la especificación inicial, como de la

descripción hardware del circuito añadido. Sin embargo, en la presente aproximación, el resultado se restringe al nivel de enumeración de tareas por partición con sus características propias.

De la misma manera, surge la simplificación acerca de la especificación inicial. En un caso real, sería el propio código del algoritmo el que debería ser analizado, y del que se extraerían todas las tareas que lo componen.

Sin embargo, y debido en parte a la dificultad y no estandarización del lenguaje de entrada, se comienza con un conjunto de tareas ya extraídas, en las que constan las distintas operaciones que se deben efectuar. En consecuencia, tampoco se realiza la fase de elección de la granularidad de una manera automática.

Así, y en resumen, el espacio de entrada estará constituido por un grupo de tareas bien definidas, especificadas por medio de un conjunto ordenado de operaciones. Estas tareas dispondrán de una serie de dependencias conocidas entre ellas, que surgen del propio código inicial de la especificación dada.

En cuanto a la restricción temporal que viene impuesta, se considera una única que afecta a la totalidad del algoritmo. Es posible que existan sistemas en los que en lugar de esa restricción global, tenga sentido aplicar un conjunto de restricciones más concretas que deban ser cumplidas por partes específicas del algoritmo.

El hecho de considerar una sola restricción viene impulsado por un deseo de simplificación y concreción, y todo lo argumentado en este trabajo es igualmente aplicable al caso de múltiples restricciones, sin ningún tipo de dificultad añadida. De esta manera, no existe ninguna complejidad conceptual que diferencie un caso del otro.

2.2.2. Especificación de la arquitectura objetivo.

Un factor clave que determina las características básicas del sistema que se va a generar es la composición de su arquitectura objetivo [BaGa97]. En los procesos de diseño de circuitos hardware por separado, o bien únicamente de programas software, siempre han existido distintas posibilidades de organizar sus componentes, lo que delimita las propiedades del objeto generado.

De esta manera, al hablar de un circuito es posible especificar qué tipo de dispositivos de almacenamiento son preferibles frente a otros, así como el modelo, número y situación de los buses que interconectan los distintos módulos.

Por otra parte, al trabajar con programas software, es normal crear un tipo de jerarquía en la que se determine cómo se deben comunicar los distintos procedimientos, y cuál es la forma de intercambiar información a través de ellos, así como la relación que se va a tener respecto al núcleo del sistema operativo utilizado.

En estas circunstancias, es posible que las características de tiempo y coste asociadas a las distintas entidades generadas varíen de un caso a otro. Esto se debe a que unas estructuras son más convenientes que otras para un determinado tipo de problemas, ya que utilizarán específicamente recursos más esenciales en su resolución, y por lo tanto, aumentarán la efectividad.

Partiendo de las explicaciones previas, se puede notar la importancia no sólo del diseño correcto de entidades digitales, sino también de la estructura que los va a sustentar, dotándolos de características y ventajas propias.

En los casos comentados anteriormente, referidos a dispositivos hardware y software, esta caracterización de su estructura tiene una

propiedad básica, y es que resulta homogénea. De esta manera, al desarrollar un circuito, se deben tener en cuenta aspectos puramente hardware, similares a los que posteriormente se van a tratar en el proceso de fabricación, y lo mismo ocurre de una forma análoga en el caso del software.

Sin embargo, en los sistemas tratados por la metodología de Codiseño estas consideraciones se complican, ya que se tienen relaciones entre los dos subsistemas que generan dependencias híbridas entre ellos. Por lo tanto, es preciso catalogar y especificar estas dependencias, lo que dará lugar a la caracterización de la arquitectura del sistema, y en consecuencia a las propiedades básicas que posea.

La característica esencial de todo sistema de Codiseño reside en la utilización de un bus para comunicar ambos subsistemas. De esta manera, y teniendo en cuenta que el núcleo del programa inicial, como ya ha sido comentado, reside en la partición software, o bien los datos iniciales, o bien datos parciales generados por esta partición han de ser transmitidos al subsistema hardware, y este, evidentemente, habrá de devolver los resultados obtenidos mediante el cálculo sobre dichos datos.

Por lo tanto, la primera característica básica que se considera en la presente aproximación al Codiseño, es la existencia de un bus físico que comunica ambos subsistemas, hardware y software, de una manera biunívoca.

En la presente aproximación, y únicamente por necesidad de simplificar, se ha considerado un sólo procesador asociado a la partición software y un único circuito para la parte hardware. Esta decisión, viene motivada por la necesidad de fijar los conceptos teóricos que aparecerán a lo largo de este trabajo sobre una base lo más sencilla posible. Esto no significa que no sean aplicables en el caso de expansión de los subsistemas, es más, dicha situación no presentaría ningún problema

conceptual extra[1], pero dificultaría a nivel formal la definición de expresiones, lo que sólo conllevaría una falta de claridad explicativa en las mismas.

Para la partición software, se ha elegido un procesador estándar de propósito general. Por las mismas razones de simplicidad, se han evitado entornos que tengan entre sus posibilidades características complejas, como la ejecución segmentada de instrucciones. Esto conllevaría una mayor dificultad en el proceso de estimación asociado a la parte software, que como se manifestó anteriormente queda al margen de la presente investigación. En consecuencia, el procesador que se utiliza en la arquitectura final es un modelo simple, con un método lineal de ejecución de instrucciones.

Para futuros trabajos relacionados con esta partición, sería interesante considerar las ventajas producidas por el uso de procesadores más complejos. Asimismo, el estudio del sistema operativo requerido para su funcionamiento también sería trascendente. Este sistema operativo tendría que estar compuesto por un núcleo mínimo, que permitiera la realización de todas las operaciones contenidas en la especificación inicial, eliminando todos los detalles innecesarios a fin de simplificar lo más posible la capacidad de almacenamiento de la partición.

En esta aproximación, se ha considerado la existencia de un núcleo de sistema operativo que sea capaz de realizar transmisiones de datos con la *partición hardware utilizando el bus del sistema.*

Otro punto muy importante es la presencia de una memoria, capaz de almacenar tanto los datos de entrada, como el código que deba ser ejecutado por el procesador, además del conjunto de valores iniciales. Esto

[1] Aunque sí añadiría una complejidad temporal al proceso de particionamiento, lo cual daría pie a subsiguientes optimizaciones de los métodos para solventar estos aspectos negativos.

contribuye a la existencia de un coste fijo de partida, que se debe tener en cuenta a la hora de diseñar el sistema.

El tamaño de la memoria que se va a utilizar debería ser estudiado para cada una de las aplicaciones sobre las que se vaya a trabajar. Por regla general, este tamaño tendría que ser lo más pequeño posible, fundamentalmente debido a dos razones: por un lado, la necesidad de que su coste sea mínimo, y por otro, la conveniencia de manejar sistemas reducidos. No en vano, la cada vez más numerosa presencia de estos sistemas hace que el espacio que ocupan tienda a ser menor, a fin de facilitar su integración en entornos más amplios.

No obstante, el ajuste excesivo del tamaño de la memoria puede acarrear efectos colaterales nada deseables. Si una vez realizado el proceso de Codiseño se llega a la conclusión de que la cantidad de datos iniciales e intermedios, junto con el conjunto de líneas de código que forman la partición software, sobrepasan el tamaño de memoria estipulado, sería necesario añadir un nuevo módulo, lo que encarecería el sistema, o bien realizar de nuevo el proceso de Codiseño de tal manera que esta solución no se vuelva a producir. Esta última opción es igualmente inconveniente, ya que el proceso completo suele consumir un tiempo elevado, que es necesario no incrementar para hacer el producto final realmente competitivo.

Por lo tanto, no se propone ningún tamaño definitivo de la memoria, pero se entiende que se va a optar por un modelo con una capacidad de almacenamiento media-baja.

Otra característica importante acerca de la memoria es su estructura. Obviamente, existen distintas alternativas, desde las más baratas y simples, a las más rápidas y complicadas de predecir. Las dos opciones que se plantean como más factibles en este tipo de sistemas es el uso de memoria con un único nivel, o bien la incorporación extra de un nivel de cache [LiMW96] [MaGA95].

Está claro que si se opta por esta segunda opción, el rendimiento del sistema aumentará, debido a las propias características de este tipo de memoria. Así, el acceso a datos muy frecuentes se desarrollaría en un tiempo mucho menor que en el caso de considerar únicamente la memoria principal.

Sin embargo, también es cierto que el coste del sistema se encarecería, lo cual no es nada deseable, según los comentarios aparecidos en secciones anteriores. Otro punto en contra es la dificultad añadida en las estimaciones de tiempo asociadas. De esta manera, sería preciso estudiar los índices de acierto de los distintos accesos, según las cadenas de referencias de datos que previsiblemente se pueden producir [LiMW95]. Por lo tanto, y siempre con deseo de simplificar[2], se propone el uso de memoria con un único nivel.

Llegados a este punto, es preciso hablar de otra función esencial que va a llevar a cabo la memoria. Previamente se comentó la necesidad de comunicación de datos entre ambas particiones, la cual se desarrollará a través del bus del sistema. No obstante, es necesario no sólo especificar el medio físico de transmisión de los datos, sino también el mecanismo que se va a utilizar para llevarlo a cabo. Es lógico que se considere alguna forma de *sincronizar las necesidades de ambas particiones, las cuales van a estar* accediendo constantemente a una serie de recursos e información comunes.

[2] La consideración de modelos de memoria más complejos afectaría directamente al proceso de estimación de parámetros software, lo cual no es un tema principal de esta investigación. Por lo tanto, esta simplificación no influye de manera directa en las propuestas realizadas a lo largo de los próximos capítulos. Es evidente que todo futuro trabajo debería considerar modelos de memoria más complejos.

De esta manera, surgen varias posibilidades acerca de cómo la partición software ha de leer un dato producido por la hardware, o cómo esta última debe enviar un resultado a la primera. Básicamente, los modelos de comunicación por los que se pueden optar son los denominados de *paso de mensajes* y de *memoria compartida*.

El primero de los mecanismos se caracteriza por la posibilidad de comunicación directa entre el software y el hardware. De esta manera, si el primero posee un dato requerido por el segundo, la transferencia se puede realizar directamente entre particiones a través del bus. Por consiguiente, este es un método rápido, pero posee el inconveniente de necesitar para su funcionamiento una lógica costosa entre los dispositivos.

No hay que olvidar la posibilidad de que exista más de un único elemento por partición, lo que conllevaría desarrollar un mecanismo en el que se indique la petición de qué datos y a qué módulo, así como los procesos de sincronización adecuados. Esto elevaría el coste del sistema, aumentando su complejidad de una manera excesiva.

La segunda posibilidad, por la cual se ha optado, es la de *memoria compartida*, que es menos eficiente, pero también mucho más sencilla de manejar y estudiar. Así, todos los datos producidos por las tareas se almacenan en posiciones de memoria determinadas. Cada vez que un dispositivo necesita uno de estos datos, sabe en qué posición puede encontrarlo, y basta con un acceso a esta dirección para que los valores estén disponibles.

Obviamente, la desventaja de este mecanismo es el peor rendimiento que ofrece, ya que por lo general, los accesos a memoria suelen llevar asociados un gran empleo de tiempo.

En esta situación, aparece una dificultad añadida, debido a la competencia por los recursos del sistema. En muchos casos, ambas particiones se verán en la necesidad de acceder a memoria, ya sea para

leer datos o para escribirlos. Como el bus es un recurso limitado, sólo uno de sus posibles clientes puede acceder a él en cada momento, por lo que sería necesario la determinación de un mecanismo de arbitraje.

Sin embargo, es posible evitar este tipo de situaciones si se limita el número de dispositivos que pueden ser maestros del bus a sólo uno. De esta manera, el procesador de la parte software, que evidentemente dispone de la lógica necesaria para acceder directamente al bus, pasaría a ser el único maestro disponible, mientras que el circuito hardware se vería supeditado a realizar todas las transferencias en modo esclavo[3].

Para el presente trabajo se ha adoptado este modelo, debido fundamentalmente a dos factores. Primero, para simplificar la lógica asociada al circuito. De esta manera, sólo es necesario el hardware apropiado para acceso al bus en modo esclavo, evitando las más complejas funcionalidades asociadas a la posibilidad de trabajo en modo maestro. Por otro lado, no se necesita el mecanismo de arbitraje del bus, que de igual manera obligaría a introducir una lógica en el circuito que permita interaccionar con el procesador de una manera eficaz.

Una última cuestión relacionada con el problema de acceso al bus, reside en el modo en que el circuito recibirá la información. Es evidente, que en determinados momentos, la partición hardware necesitará una serie de datos para efectuar las operaciones asociadas a una cierta tarea, datos estos que, en muchas ocasiones, habrán sido producidos por la partición software.

Por lo tanto, es imprescindible proveer al hardware de esta información requerida a fin de que continúe sus operaciones sin introducir ningún retraso en el sistema. Con el modelo adoptado, en el que el procesador es el único

[3] En el caso de múltiples procesadores software habría que determinar cuál de ellos realizaría la función de maestro, teniendo en cuenta posibles criterios de rendimiento.

maestro posible, se presentan dos alternativas a la hora de realizar la transferencia de datos.

La primera, se basa en la propia naturaleza de los sistemas que se van a sintetizar. En estos, una vez finalizado el proceso de Codiseño, con la partición hardware y la software perfectamente especificadas, el esquema temporal de comienzo y finalización de todas las tareas está completamente fijado. Esta labor se ha llevado a cabo como punto necesario para la estimación del tiempo global del sistema, y por lo tanto es perfectamente conocida.

Con esta consideración, el sistema es consciente desde el primer momento de su funcionamiento y de la distribución de tiempos asociada a la planificación de tareas de una manera inequívoca [BeEO95]. Esta relación de tiempos ha de mantenerse constante, ya que se descarta la realización de una planificación dinámica[4] [BLMS98] [MoDe97], por lo que el procesador conoce en cada momento los instantes de tiempo en los que el circuito hardware necesita o ha producido unos datos en particular.

De esta manera, sería el propio procesador el que se encargaría de realizar la transferencia de datos a través del bus, bien sea de lectura o de escritura. En consecuencia, el circuito hardware encontraría la información disponible en el instante apropiado, o sencillamente volcaría sus resultados al bus cuando los hubiera procesado, sin tener que realizar ninguna interrupción a la parte software.

Una desventaja de esta propuesta reside en la excesiva carga que se añade al procesador, ya que no sólo debe realizar la tareas que le haya

[4] Proceso que consiste en realizar una nueva distribución temporal de tareas una vez que el sistema ha comenzado su ejecución, atendiendo a diversos motivos de conveniencia dados por la situación general de los procesos.

asignado el particionador, sino que además le corresponde la labor de efectuar las comunicaciones de datos pertinentes con la otra partición.

Pero la característica más problemática está asociada a la imposibilidad de predecir con exactitud los datos de entrada. Si se ha estimado un tiempo de finalización de una tarea, y debido a la particularidad de los operandos se requiere más margen para finalizarla, se produciría un desfase entre el componente hardware y el software.

En consecuencia, si el procesador decidiera recoger los resultados generados por el circuito en el instante establecido, adquiriría unos datos erróneos, lo cual repercutiría en el mal funcionamiento de todo el sistema. Este modelo sólo sería conveniente para entornos con unos valores de entrada determinados e invariables.

Por lo tanto, se precisa una sincronización adecuada entre ambos componentes. Así, la segunda opción, que es la que se propone para este trabajo, consiste en que, al no poder acceder el hardware a la memoria por sí solo, debe informar de su necesidad al procesador, mediante una línea de interrupción. De esta manera, el maestro conoce la petición realizada sobre unos datos particulares, encargándose de su lectura de la memoria y de su volcado al bus.

Una vez realizada esta operación, avisa al hardware de ese hecho, pudiendo este último recogerlos mediante un acceso en modo esclavo. La condición de que los datos se encuentran listos podría venir dada por la decodificación de unos ciertos bits del bus de direcciones, que servirían de capacitación al circuito de una forma inequívoca.

Una desventaja de este método reside en la necesidad de dotar al hardware del mecanismo necesario de petición de interrupciones, para que pueda realizar la solicitud de datos en todo momento. Esto, por supuesto, conlleva un incremento del coste al aumentar la complejidad, así como también del tiempo de interacción entre las dos particiones. Sin embargo, el

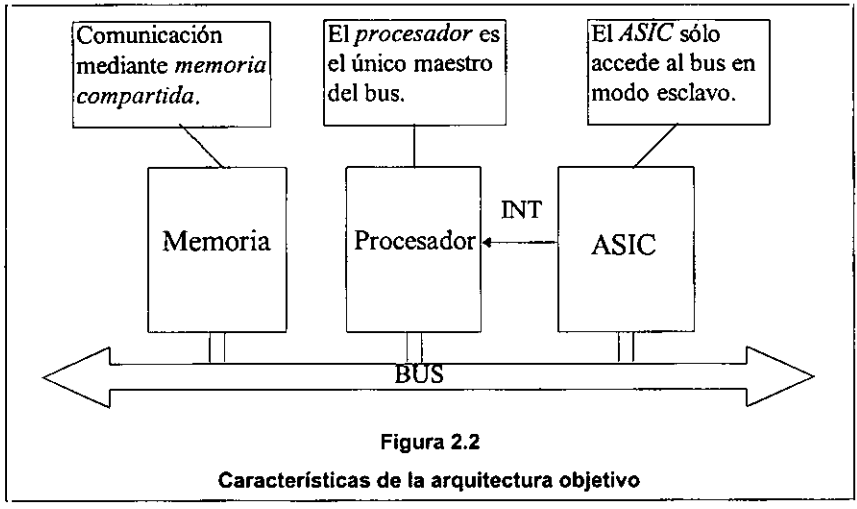


Figura 2.2

Características de la arquitectura objetivo

procesador, al ser estándar, debe disponer de unos procedimientos reglados de recepción y servicio de interrupciones, sin coste adicional.

En resumen, y según todo lo expuesto en este capítulo, la arquitectura tipo que propongo para la realización del proceso de Codiseño (Figura 2.2), siempre teniendo presente la necesidad de un estudio ulterior más concreto del problema particular, vendría caracterizada por los siguientes puntos:

- Un entorno que proporciona un coste fijo inicial, formado por el procesador estándar, la memoria y el bus del sistema.
- Un circuito hardware, que ha de cumplir que su coste sea mínimo para la restricción temporal impuesta, coste este que se debe sumar al inicial para calcular el coste total del sistema.
- Un modelo de comunicación de parámetros por memoria compartida, en el que el procesador es el único maestro posible, y por lo tanto, el circuito hardware ha de acceder a la información en modo esclavo.
- Una política de transmisión, basada en una petición de datos por parte del circuito al procesador, mediante un mecanismo de interrupciones.

Una vez planteada esta arquitectura, y debido a que el presente trabajo se centra en el proceso de estimación de parámetros asociados a la

partición hardware, es preciso explicitar qué simplificaciones teóricas se han tomado a la hora de tratar el subsistema software.

2.2.3. Consideraciones acerca de la parte software.

Es normal que todo trabajo desarrollado sobre una metodología tan amplia y ramificada como es el Codiseño, se centre especialmente en una serie de aspectos concretos que sean relevantes para apoyar las ideas que se quieren transmitir.

La mayoría de las veces, el hecho de pretender tratar todas las facetas interesantes puede no ser una decisión conveniente. Una de las mayores virtudes de la investigación debería ser la de dejar líneas abiertas para futuras expansiones, ya que de lo contrario se cae en el riesgo de crear un trabajo cerrado en apariencia, pero que sin embargo tenga carencias debido al amplio espectro de trabajo abarcado.

En este caso, se ha preferido no entrar en detalle en los métodos de estimación de parámetros software, por lo que se ha realizado una sustitución de metodologías reales por un modelo simplificado de este aspecto.

Primeramente, es preciso decir que existen diversos trabajos acerca de cómo manipular dichos parámetros software [GuDe97] [Gupt95]. Estos abarcan desde estimaciones con sistemas de memoria de un sólo nivel, hasta modelos más complejos con memoria cache. En este último caso, la elección del dispositivo de almacenamiento es fundamental a la hora de calcular el tiempo de transferencia de las distintas instrucciones y datos al procesador, y por lo tanto, del tiempo global consumido por las tareas.

En este tipo de aproximaciones, una labor fundamental es la estimación del número de veces que se ejecutarán los distintos bucles o ramas condicionales existentes [Leup99], lo que está muy unido al cálculo de los aciertos de cache dependiendo de las distintas referencias realizadas.

Estos procesos son bastante complicados, y se basan en métodos de predicción estática del flujo de los datos, en la determinación de caminos mutuamente exclusivos y en otras técnicas similares [WuLa94].

Muchas veces, cuando se requieren grados de exactitud muy elevados, se recurre al análisis dinámico, en el que se utilizan cargas de datos que se suponen estándar, y se tratan mediante el código de las tareas contenidas en el procesador. De esta manera, se pueden calcular datos acerca de porcentajes en toma de decisiones y del número medio de ejecución de los distintos bucles.

Conjugando ambos métodos, el análisis estático y el dinámico, es posible alcanzar conclusiones más precisas acerca de los tiempos de ejecución de las distintas tareas software, evidentemente, sujetas al modelo de memoria elegido.

Sin embargo, muchas veces no es posible realizar análisis dinámicos, ya que puede ser que no se disponga de la arquitectura objetivo. En ese caso, el uso de técnicas de simulación y prototipado puede resultar de gran ayuda en los procesos de estimación.

Esto, que por supuesto no debe olvidarse dentro de un estudio de Codiseño, y que debería mejorarse con nuevas investigaciones, ha sido obviado en el presente trabajo. Por lo tanto, los parámetros característicos de las tareas asociadas a la partición software han sido presupuestos, y por lo tanto no medidos, utilizando para ello un modelo de generación basado en valores razonables.

Primeramente, el parámetro más característico relacionado con las tareas software es su tiempo de ejecución. Esto se debe a la especial lentitud de este subsistema, que induce a la presencia de numerosos cuellos

de botella en el diseño[5]. Por lo tanto, los tiempos de las tareas software deben ser siempre bastante mayores que su equivalente en hardware.

Para cada tarea, se propone una relación entre estos tiempos en función de su previsible complejidad, teniendo en cuenta la carga operacional que posea y el posible grado de paralelismo de las operaciones (a mayor número de estas, y con mayor grado de paralelismo, mayor desventaja tiene el software, debido a su ejecución obligatoriamente secuencial. Véase el Apéndice A para más detalles acerca de la estimación de tiempos software).

Por supuesto, el sistema de Codiseño presentado es totalmente modular, y *permite la incorporación de cualquier otro método de estimación de tiempos más elaborado para este subsistema.*

Aparte de este parámetro, también es preciso determinar cuál es el coste asociado a cada una de las tareas de esta partición. Es obvio que para la parte hardware este concepto tiene pleno sentido: el coste es directamente proporcional al número de unidades funcionales que hay que añadir al sistema. Esto es un factor tangible, que afecta decisivamente al proceso de fabricación.

Sin embargo, el coste asociado a la parte software carece de este sentido tan real. Para ejecutar el subsistema asociado es preciso disponer de un procesador estándar y de un módulo de memoria, que evidentemente tienen un coste relacionado. Este valor, junto con el del bus, conforman el coste inicial de todo sistema, que es obligatorio asumir en una etapa previa al comienzo del Codiseño.

[5] Es obvio que esto ocurre, porque en caso contrario no tendría sentido la utilización de la metodología de Codiseño, ya que no existiría ningún problema para cumplir la restricción temporal.

Genéricamente, cuando en esta metodología se habla de coste, este parámetro excluye la cantidad inicial mencionada, ya que esta no puede ser minimizada durante el proceso de particionamiento.

Por lo tanto, si se descuenta el coste inicial del sistema, el asociado estrictamente a la partición software resulta ser nulo. En efecto, el código como tal no tiene una existencia física, y por lo tanto no importa a estos efectos que todas las tareas estén contenidas en la memoria del sistema, o que la mayoría de ellas se encuentren planificadas en hardware. El coste software en ambas situaciones sería equivalente e igual a cero, siempre que la memoria del sistema no se desborde.

En este trabajo no se ha considerado la posible restricción introducida por el tamaño de la memoria. En el caso de que esto fuese necesario, no se presentaría ningún problema conceptual añadido. Así, sería sencillo programar el particionador para que rechazase cualquier solución parcial en la que el tamaño de las tareas software sobrepasase la memoria del sistema.

De esta manera, y con las consideraciones previas, el modelo de simplificación del subsistema software queda completo.

2.3 Tratabilidad y Aplicabilidad.

Una vez comentadas las características de la arquitectura objetivo elegida, así como las simplificaciones llevadas a cabo, es preciso realizar un estudio acerca de si este entorno de Codiseño es apropiado para su aplicación a los distintos problemas y la versatilidad que posee.

De esta manera, en la presente sección, comentaré la facilidad y potencia de uso del sistema (*tratabilidad*), junto con las posibilidades de ser usado sobre problemas reales (*aplicabilidad*).

2.3.1. Tratabilidad.

Todas las simplificaciones introducidas en el sistema tienen dos objetivos claros. Por un lado, hacer que el núcleo teórico que sustenta a la metodología de Codiseño sea fácilmente comprendido, sin que se vea afectado por otras complejidades no conceptuales provenientes de un entorno de trabajo excesivamente recargado[6].

Por otra parte, evitar que este entorno se convierta en un factor contraproducente, ya que si bien es posible dotarlo de una gran cantidad de características avanzadas, este hecho puede volverse negativo en el caso de que las técnicas de estimación alcancen tal nivel de complejidad que no sea posible abordarlas eficazmente.

No hay que olvidar que el proceso de Codiseño ha de ser fácil de tratar, y que no debe extenderse demasiado en el tiempo. Las últimas tendencias imponen una reducción máxima del tiempo de diseño de los circuitos, y cuanto más directas y rápidas sean las metodologías utilizadas, mayor valorización final del producto conseguido.

De esta manera, la incorporación de características avanzadas, como un sistema de memoria cache o procesadores segmentados, puede llevar a complicar excesivamente las ya de por sí arduas tareas de Codiseño.

Así, la estimación del tiempo de ejecución de las tareas en el procesador, bajo las circunstancias anteriores, es mucho más complicada y sujeta a error. La predicción de fallos que se produzcan en las referencias a memoria no es tan fiable como en el caso de arquitecturas más sencillas.

---

[6] Es preciso recalcar que no se pretende la generación de un sistema de Codiseño completo, sino el estudio de ciertas tareas esenciales a las que posteriormente se podrán añadir nuevas consideraciones.

Esto redundaría en unas dificultades añadidas, lo que sin duda complicaría el estudio y verificación de las técnicas de estimación y particionamiento presentadas en este trabajo.

Sin embargo, todas estas innovaciones afectarían fundamentalmente a los mecanismos de la partición software, con lo que las metodologías presentadas en este trabajo no se verían directamente afectadas por estas variaciones.

No obstante, el modelo de arquitectura ofrecido ha de estar también propuesto pensando en la influencia que pueda tener sobre este subsistema, aunque ahora se presente un modelo simplificado del mismo, con vistas fundamentalmente a futuras ampliaciones.

En estas circunstancias, y como no existe una razón de peso, se ha optado por el modelo de arquitectura simple, en beneficio de esta tratabilidad necesaria para el buen desarrollo del sistema. Esto no quiere decir que las arquitecturas complejas no deban abordarse, todo lo contrario, sino que esa introducción ha de ser muy progresiva, para que los cambios producidos no afecten en exceso a la facilidad de desarrollo del proceso de Codiseño.

2.3.2. Aplicabilidad.

Otro punto aparte que se debe considerar es si las simplificaciones introducidas, que obviamente facilitan la tratabilidad, son compatibles con las necesidades que poseen los sistemas reales. En otras palabras, si el modelo de arquitectura presentado es capaz de sustentar las operaciones exigidas por estos problemas reales, o por el contrario son excesivas para su tratamiento.

Para encontrar la respuesta a esta duda, es necesario observar las características de los sistemas que se diseñan hoy en día. Si bien es cierto que la síntesis exclusiva de circuitos hardware se deriva cada vez más hacia

un automatismo, en el caso de sistemas híbridos muchas de las tareas claves, como el particionamiento, se siguen desarrollando con la total intervención del diseñador.

Por lo tanto, se tiende al uso de arquitecturas simplificadas, no sólo por su mayor facilidad de uso, sino porque es lo que ha funcionado bien hasta ahora. Cifras recientes, indican que el mercado actual de los sistema empotrados sigue basado en microcontroladores de 16 bits, con ejecución de instrucciones totalmente secuencial, y sistema de memoria simplificado. El hecho de no introducir mejoras substanciales en estos campos es debido a que las exigencias de los problemas actuales pueden ser tratados con este tipo de arquitecturas.

Por supuesto, es obvio que estas mejoras deberán ser añadidas progresivamente, y las metodologías asociadas tendrán que ser adaptadas. Pero en el momento actual, cuando el diseño automático de sistemas aún no está muy implantado, no parece conveniente la incorporación de excesivas complejidades en el proceso.

No hay que olvidar, que en la mayoría de los casos en que ciertos algoritmos software no cumplen las restricciones temporales impuestas, el usuario afectado ni siquiera contempla la posibilidad de incorporar un circuito hardware a su sistema con el propósito de acelerarlo, sino que se limita a realizar optimizaciones sobre el código a fin de que sea más rápido, o a adquirir nuevos procesadores con mayor potencia de cálculo.

El hecho de tener que modificar la arquitectura, añadir un bus, y controlar un sistema híbrido, hace que el cambio a metodologías de Codiseño, realmente beneficiosas para solventar estos problemas y muchas veces las únicas con garantías de éxito, sea contemplado de forma recelosa, a no ser que estos cambios sean mínimos y asumibles desde un primer momento.

Es cierto que siempre hay un sector de los sistemas digitales que se caracteriza por sus restricciones absolutamente estrictas, y que necesitan de una gran potencia de cálculo y de manipulación de datos. Pueden ser, por ejemplo, los sistema digitales destinados a la transmisión y recepción de imágenes en tiempo real. Estos sistemas llevan asociados unos procesos de codificación y decodificación de datos muy exigentes, en los que el ritmo de absorción de datos es realmente alto.

En estas circunstancias, es obvio que se necesitan las más avanzadas arquitecturas [BaGa97], altamente especializadas. Así, la rapidez en el acceso a memoria, la ejecución paralela software [DeSh98] [Lin98] y la presencia hardware de módulos de proceso muy especializados son condiciones necesarias para que el sistema generado funcione de una forma óptima.

Por lo tanto, y aunque las arquitecturas simplificadas, como la presentada en este capítulo, sean todavía aplicables a la mayoría de los problemas existentes, se deben ir proponiendo las mejoras comentadas para que, progresivamente, los problemas complejos vayan siendo tratables de forma automática.

La aproximación Macroscópica: Carencias del enfoque clásico.

3

En los capítulos anteriores, se han expuesto las características básicas que definen el Codiseño, las investigaciones previas más importantes y la propuesta de entorno de diseño para el presente trabajo. Se ha visto claramente que ninguna de las aproximaciones existentes en la actualidad reúne el conjunto de consideraciones necesarias para abordar un problema con una cierta entidad de forma realista. Estas carencias se resumían en una percepción demasiado simple de los diseños hardware, en los que no se tenía en cuenta la posibilidad de reuso entre las distintas funcionalidades, o su capacidad de ejecución paralela, ni mucho menos la posible exploración del equilibrio entre estos dos factores.

Sin embargo, aunque he criticado dichas restricciones, todavía no he expuesto el porqué de ellas. En otras palabras, las carencias mencionadas son claramente negativas, y fácilmente detectables por cualquier observador externo. De esta manera, cabe preguntarse por qué no han sido abordadas por la mayoría de los sistemas existentes.

3.1 Aproximaciones clásicas: la herencia de la SAN.

Hay que recordar que el Codiseño surgió como una evolución lógica de la Síntesis de Alto Nivel (SAN). Esto se refiere al hecho de que el paso

natural después del diseño de circuitos hardware es la consideración del diseño de sistemas, siendo un sistema una realidad más compleja, en este caso formada por un componente software y otro hardware (§1). Así, parece claro que la nueva disciplina se puede considerar una expansión de un ente conocido (el circuito hardware), al que se añade una serie de nuevos elementos.

Debido a esto, la forma de abordar el problema en un principio tendía al uso de metodologías y algoritmos clásicos de la bien conocida Síntesis de Alto Nivel (Véase Figura 3.1). Antes de continuar, es preciso dejar bien claro que no critico en absoluto el mencionado hecho. Es más, creo que la aproximación a un nuevo campo a partir de otro conocido es una técnica necesaria, ya que la *herencia* de conocimientos siempre se ha mostrado fructífera en amplios ámbitos de investigación.

Sin embargo, cuando el nuevo problema presenta características intrínsecas excluyentes[1] respecto a la disciplina conocida, es preciso transformar los principios heredados para su correcta utilización. Este concepto de *transformación* no debe estar relacionado a una mera extrapolación, sino al hecho de replantearse la forma de obtener unos mismos resultados variando el entorno del problema.

Para centrar esta discusión en el presente campo de investigación, compararé los requerimientos de las metodologías clásicas de la SAN con las necesidades propias de los problemas de Codiseño.

[1] Por *excluyentes* me refiero a aquellas particularidades de un problema que no pueden ser tratadas por metodologías asociadas a otro. Así, un sistema en tiempo real no puede ser diseñado con una técnica para resolver problemas sin restricciones temporales.

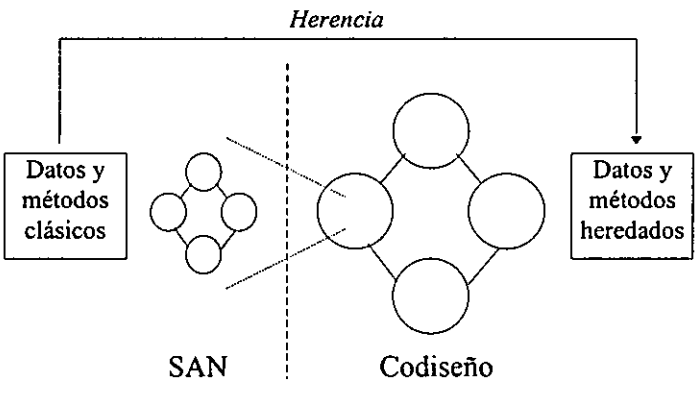


Figura 3.1
Herencia de las metodologías de la SAN

3.1.1. Metodologías clásicas de la SAN.

En un principio, la Síntesis de Alto Nivel se caracterizó por tener como objeto de trabajo circuitos de una complejidad media. Es lógico que esto sea así, ya que los diseños que tradicionalmente se han tratado de automatizar son aquellos con una mayor producción de mercado, y la mayoría de las veces, una no muy alta complejidad.

Este hecho marca definitivamente qué procesos son racionalmente aplicables a esta disciplina, ya que el tiempo de diseño, factor directamente proporcional a la complejidad del problema, es la restricción fundamental en la mayoría de las actividades automáticas.

Si se repasan las tareas básicas que se llevan a cabo en la SAN, se puede llegar a la conclusión de que las dos más críticas, que además están relacionadas de una forma interdependiente, son:

- Planificación de operaciones en los distintos ciclos de reloj (crítica para el tiempo de ejecución).

- Asignación de componentes hardware a cada una de las operaciones (crítica para el coste del sistema).

Se ve que aquí aparecen de nuevo los dos factores, siempre opuestos, de tiempo y coste. En esta situación, normalmente se ha de cumplir una restricción impuesta asociada a uno de los dos parámetros mencionados, mientras que el otro de ellos debe ser minimizado. A la vista de estas circunstancias, ambas tareas se reducen a problemas de optimización.

La característica básica de estos problemas es su alto tiempo de proceso, principalmente debido al amplio espacio de diseño que es necesario explorar. De esta manera, la cualidad de óptimo introduce una carga temporal muy fuerte en el sistema. Si se relaja esta condición, y sencillamente se buscan soluciones suficientemente buenas, habitualmente se obtienen peores resultados, a veces con una calidad alejada de la esperada.

Para llevar a cabo esta optimización existen innumerables algoritmos ampliamente probados. Estos, varían entre los más mecánicos y precisos, como la búsqueda exhaustiva y la *ILP*, hasta los más heurísticos, como la planificación basada en listas, en fuerzas, o en *branch-and-bound*. Sin embargo, todos ellos tienen la característica mencionada anteriormente: o poseen una alta complejidad, o su eficacia se reduce de una manera drástica.

Con esto se ha querido dar una idea clara, no de las operaciones en concreto de la SAN, sino de su complejidad temporal, aun para problemas de reducido tamaño. La última observación iría orientada a señalar que la unidad básica de tratamiento en esta disciplina es la *operación*, definida como la tarea capaz de ser implementada por una unidad funcional. Además, el número de ellas estaría determinantemente relacionado con el tiempo de ejecución final del diseño automático.

3.1.2. Las nuevas tareas del Codiseño.

Como punto de partida, habría que dejar claro que cada uno de los módulos o funcionalidades que componen un problema de Codiseño, y que han sido entresacados de la descripción inicial por medio de la elección de la granularidad, corresponde a uno de los circuitos SAN comentados en la sección anterior. Desde este punto de vista, los nuevos problemas sí son una expansión lógica respecto de los antiguos.

Así, cabe esperar que la complejidad asociada a la tarea de decidir cuál es la implementación óptima para la parte hardware del sistema, se vea incrementada en función del número de dichos módulos con que cuente el problema.

Sin embargo, no es únicamente el proceso de estimación el que consume la mayor parte del tiempo, sino que existe la necesidad de realizar un particionamiento con el fin de llegar a la mejor solución final. Aunque más adelante se expondrán con detalle las características de este particionamiento (§6), comentaré ahora las singularidades esenciales, para comprender la verdadera complejidad de todo el proceso.

Un algoritmo de particionamiento estándar, como puede ser cualquiera de los iterativos, tiene una complejidad temporal que se puede acotar por $N^2 \cdot log(N)$, siendo N el número de tareas del problema. Esta expresión correspondería al número total de pasos que se requieren para alcanzar la solución final.

Si se define cada paso del algoritmo como el conjunto de operaciones por las cuales se decide mover temporalmente un nodo a una partición determinada, se obtiene que cada uno de estos pasos está formado por:

- Determinación del nodo que se moverá a la partición opuesta.
- Si la partición destino es la hardware, determinación de la implementación que se utilizará para el nodo escogido.

- Si la partición destino es la hardware, determinación de la posición relativa del nodo dentro de su rango de movilidad, y relación con el resto de nodos que formen la partición, lo que configurará la posibilidad de ejecución paralela.

Posteriormente, habrá que repetir todas las operaciones anteriores para cada uno de los nodos que haya en el sistema. Con esto, se obtendrá una estructura definida de las dos particiones existentes. Sin embargo, es claro que no se puede garantizar que los resultados hallados sean los óptimos, y por lo tanto, la posibilidad de que la solución devuelta corresponda a un mínimo local es alta. En este punto es preciso comenzar de nuevo la totalidad de los pasos, con unas condiciones iniciales distintas, a fin de crear un mecanismo capaz de escapar de dichos mínimos locales. Este mecanismo varía según el algoritmo concreto que se use, aunque la mayoría tiene en común el utilizar la mejor solución obtenida a la finalización del algoritmo como partición inicial de una nueva ejecución.

Se ha escogido un tipo iterativo de algoritmo de particionamiento para describir sus pasos por ser de los más extendidos, aunque cualquier otro modelo estándar, como los basados en la aleatoriedad (*Simulated Annealing* [KiGV83], Genéticos [DiJh97], etc.), tiene una estructura similar.

Enumerando todas estas tareas, que luego explicaré con más detalle en el capítulo correspondiente, he querido ofrecer la visión de un proceso complejo, el cual conforma el núcleo del Codiseño.

Esta complejidad se ve drásticamente aumentada cuando se observa que para realizar cada una de las operaciones descritas, es necesario obtener una medida de la calidad de cada solución parcial, que será comparada con las del resto para realizar cualquier tipo de decisión. Y la obtención de ese factor de calidad se produce mediante la realización del proceso de estimación, que corresponde al cálculo de área y tiempo, y que

tradicionalmente se ha venido realizando mediante las operaciones explicadas en la sección anterior.

Por lo tanto, se ve que el complejo mecanismo de planificación y asignación de unidades funcionales, que por sí solo consumía una gran cantidad de tiempo en la SAN, ahora se tiene que repetir infinidad de veces dentro del proceso de particionamiento en Codiseño, para obtener la calidad relativa de un diseño dado. No sólo eso, sino que ahora además la complejidad es mucho mayor: antes se estimaba un circuito hardware equivalente a un módulo de Codiseño; ahora, la partición hardware está formada por un conjunto de estos módulos entremezclados, que puede ser tan grande como el número de tareas totales del sistema.

La conclusión que se pretende transmitir en este punto es que si, como ya se ha mencionado, la SAN trataba problemas de una complejidad media, los nuevos problemas asociados al Codiseño, y basados en técnicas heredadas, no pueden sino ser muy restringidos y limitados en su composición.

Dicho de otra manera, para que por medio de estas técnicas sea factible realizar un proceso de Codiseño, los problemas tratados han de tender a los clásicos de la SAN, que conformaban el límite de complejidad *más allá de los cuales la metodología se hace inviable.*

Si el número de módulos es muy pequeño, o cada uno de los módulos está formado por un conjunto limitado de operaciones, el proceso de particionamiento se simplifica enormemente (hasta el punto de ser un problema obvio, fácil de ser tratado de una forma manual), y el Codiseño sí pasa a ser una actividad viable.

Otra de las posibilidades usadas hasta ahora al trabajar con problemas de Codiseño es no reducir la complejidad del problema en sí, es decir, considerar casos de mayor entidad, pero simplificando el modelo utilizado para hallar la solución. De esta manera, es posible ver aproximaciones que

tratan problemas con un alto número de módulos, cada uno de ellos formado por un número elevado de operaciones, pero que a la hora de proponer un método de estimación de las soluciones, se obvian características básicas.

Aquí es donde estarían englobados aquellos sistemas que adoptan el conjunto de restricciones mencionado anteriormente. De esta manera, no consideran la posibilidad de reuso de hardware entre los distintos módulos, no tienen en cuenta la capacidad de ejecución paralela entre las particiones hardware y software ni entre las distintas tareas de la partición hardware, y sólo admiten la posibilidad de una única implementación hardware para cada módulo.

Aunque las limitaciones anteriores puedan no causar sorpresa, debido a su gran proliferación en la literatura, hay que decir que su adopción supone una gran pérdida del nexo que todo modelo de sistema ha de tener con la realidad.

De esta manera, y teniendo presente lo expuesto en párrafos anteriores, se puede llegar a la siguiente conclusión: si se consideran problemas de una complejidad alta *y* se tratan de resolver con un modelo real, el problema no es factible en términos del tiempo de proceso.

En este punto, es donde hay que proponer nuevas metodologías que sean capaces de hacer frente a las situaciones propias de Codiseño. Por supuesto que en una primera instancia se tendrá que trabajar con modelos simplificados, como siempre que se trate de introducir una novedad, pero siempre con la idea de una posterior e inmediata expansión, hasta finalizar con un grado de realismo aceptable a las necesidades de los problemas de hoy en día.

3.2 La importancia del grado de abstracción.

De la lectura de las secciones previas cabe extraer la conclusión de que es preciso abordar el problema de Codiseño desde un nuevo punto de vista, para solventar el inconveniente del alto tiempo de proceso.

Sin embargo, parece ser esta una labor difícil habida cuenta de la gran cantidad de técnicas existentes en la SAN, todas ellas con una probada eficacia a lo largo de varios años. Puede surgir la duda de que *cómo se van a mejorar las metodologías en términos de tiempo, después de tantas optimizaciones y extensivas pruebas*. Realmente, si de veras existiera la posibilidad de esta mejora, ya habría sido propuesta hace tiempo.

Estos argumentos, aun siendo perfectamente válidos, esconden una sutileza que impiden ver el camino que es preciso seguir. Efectivamente, las técnicas existentes obtienen magníficos resultados, y además, están bastante optimizadas en cuanto a su rendimiento. Por lo tanto, si se parte de la base de que dichas técnicas son, *a priori*, difíciles de mejorar, y el espacio de los problemas por resolver debería ser irreducible, como se ha comentado anteriormente, surge una pregunta al parecer sin respuesta: ¿Dónde cabe la tan esperada innovación para hacer frente a las nuevas exigencias de diseño?. La respuesta, aunque pueda parecer subjetiva, reside en la consideración del *punto de vista* adecuado.

Así es, en mi opinión, es este punto de vista, o en términos más científicos, el *grado de abstracción*, lo que sería necesario adaptar para dar un paso definitivo en el entorno del diseño automático, y empezar realmente a tratar problemas de otra entidad.

En este punto, lo primero sería definir qué se está entendiendo al mencionar el concepto de grado de abstracción. De un modo formal, se podría catalogar como *el conjunto de entidades de un problema asociadas a*

un cierto nivel de detalle, junto con las técnicas de trabajo apropiadas para producir un resultado invariante.

Quizás la condición de *resultado invariante* es la que marca de una manera más definitiva la relación de grados de abstracción. Este hecho se refiere a que un mismo problema puede ser abordado desde distintos puntos de vista, y todos ellos, si son correctos, deberían llegar al mismo resultado, la solución del problema. Y es precisamente aquel punto de vista que sea capaz de conducir a dicha solución de la manera más directa e intuitiva, el grado de abstracción adecuado para el problema.

Si por ejemplo se está tratando de analizar el comportamiento de un circuito lógico, lo más conveniente sería realizar un estudio de sus módulos, y aplicar las funciones de conmutación apropiadas y en el orden correcto, hasta encontrar el valor del resultado producido. Sin embargo, la tipología asociada a un componente y su función de conmutación relacionada son puras entelequias, ideadas para simplificar la tarea de resolver un problema, en este caso, el análisis de un circuito.

La misma solución se debería obtener si se contempla el problema desde un grado de abstracción menor, como el electrónico. En este caso, se podría haber optado por calcular funciones de transferencia y características de transistores, y es bien seguro que se hubiera llegado a un resultado equivalente al anterior. Sin embargo, esta opción es rechazada de inmediato si se puede contar con una resolución a nivel lógico. Las razones para ello son dos: el primer método propuesto es más intuitivo y cercano al usuario, y además es mucho más rápido.

Quisiera enfatizar un poco más estos dos motivos, ya que son los principales impulsores de un aumento en el grado de abstracción, sea cual fuere el campo de aplicación en el que nos encontremos. Por un lado, la proximidad al modo de razonamiento del ser humano hace que la búsqueda de la solución del problema sea más directa y cercana. Por otra parte, el

considerar magnitudes más abstractas, y por lo tanto con más contenido en términos conceptuales, hace que la densidad de información sea mucho mayor, y por lo tanto, el tiempo de proceso tienda a disminuir.

Una consideración importante acerca de los niveles de abstracción es su versatilidad. Siempre se puede aumentar o disminuir el grado de detalle para hacer frente a un problema dado. Así, en el ejemplo anterior del análisis de circuitos, se podría haber descendido incluso al nivel de semiconductores, calculando las corrientes de partículas dentro de los dispositivos. Es bien claro que en ningún caso habríamos optado por este sistema, a no ser que el objeto del problema en cuestión hubiese sido el cálculo de características físicas del circuito a ese nivel de detalle. Entonces este método hubiese sido el apropiado, ya que los objetos sobre los que trabaja estarían directamente relacionados con el propósito del problema.

Existen innumerables casos como el expuesto anteriormente, en el que el aumento del grado de abstracción ha sido proporcional a la evolución de las técnicas de trabajo. Otro buen ejemplo de ello sería el de la programación. No hace falta recordar la evolución de los métodos utilizados en los años 50 hasta llegar a las modernas técnicas de encapsulamiento y abstracción usadas hoy en día. La aparición continua de sistemas informáticos más complejos, hace necesaria la constante evolución de los mecanismos de programación para controlarlos. A nadie se le hubiera ocurrido proponer que, como el lenguaje ensamblador es difícil de manipular en sistemas complejos, estos sistemas se deberían reducir para poder ser controlados de una manera más fácil, o simplificarlos y dejar de usar parte de su potencia de funcionamiento. Lo más lógico fue, y sigue siendo, crear entornos y técnicas de trabajo más abstractas, para hacer frente a la demanda surgida.

Si esta discusión previa parece totalmente lógica, lo mismo debería aplicarse al campo del diseño automático: las técnicas clásicas son

perfectamente asumibles, pero la dimensión que están tomando los problemas actuales hace necesaria su evolución y transformación paulatina, con vistas a hacer esta tarea más eficiente.

3.2.1. División de los Estadios del Conocimiento.

La existencia de distintos niveles de abstracción en el modelo de conocimiento no es nueva. No en vano, desde que el ser humano es consciente de su razonamiento, se produce una separación del mundo real existente respecto a su asociación con un modelo ideal mental.

Pensemos por ejemplo en las Matemáticas, cuyo objeto de estudio es un conjunto de entes abstractos totalmente inexistentes. Sin embargo, la traslación de ellos sobre el universo real genera instancias comunes e identificables, por lo que este mundo, ajeno a principios formales, se puede asemejar a la *sombra* proyectada por un universo de ideas subyacentes. Por lo tanto, mediante esta relación biunívoca, se podría afirmar que los objetos matemáticos poseen una existencia *sui generis*.

De esta manera, su manipulación produciría resultados extrapolables a problemas reales. Sin embargo, a nadie se le ocurriría utilizar los propios objetos físicos para crear entre ellos relaciones, deducciones o derivaciones, sencillamente porque ese grado de abstracción nulo, que se corresponde al universo real, no tiene asociado ninguna metodología deductiva aplicable.

En efecto, si al mundo físico se le asocian técnicas de experimentación y observabilidad, es en el universo abstracto donde se pueden aplicar los mecanismo de deducción e interpretación simbólica. Con esto se llega a la importante conclusión de que en cada nivel existen un conjunto de técnicas no extrapolables al resto de ellos, si bien todos poseen la característica de ofrecer el mismo resultado para un problema común. Así, la trayectoria de un cuerpo en movimiento podría ser hallada mediante cálculo simbólico matemático, o como observación tras un proceso de experimentación.

Corresponde al ser humano que resuelve el problema elegir el método más apropiado según la situación en la que se encuentre.

Aunque la discusión previa pueda parecer un poco *abstracta*, no en vano es este término el objeto de estudio de este capítulo, la creo necesaria para introducir situaciones más próximas a la presente discusión. Fue a mediados de los años 50, cuando, tras un fuerte avance de la teoría matemática impulsada por los avances físicos en el campo relativista, se empezaron a sentar las bases de lo que actualmente conocemos como Informática moderna.

Y fue precisamente el desarrollo del conocimiento más puro el que dio origen a esta disciplina, como una evolución natural de lo que *a posteriori* se ha conocido como Inteligencia Artificial. Es en este campo donde tuvieron lugar todas las ideas de separación de niveles de abstracción que he comentado hasta ahora, y que sirven de base a la aproximación que propongo. En este punto, no ocultaré mi influencia por los autores clásicos de la teoría del conocimiento [MDBD95]. Aunque su ámbito de acción queda algo alejado del campo de esta investigación, me gustaría realizar un inciso sobre ello, con el fin de comprender no sólo la forma, sino también el fondo del porqué de este trabajo.

Clásicamente, la Inteligencia Artificial, como método de resolución de problemas de forma automática, se estudió desde el punto de vista *conexionista*. Esto es, se intentaba replicar el mecanismo de deducción humano sobre *cerebros* electrónicos diseñados a partir de componentes físicos. Como se puede comprender, el grado de abstracción relacionado a este proceso es prácticamente nulo. Las situaciones que se podían abordar entonces eran más bien limitadas, debido a la poca capacidad del sistema considerado, y en la mayoría de los casos se tendía a simplificar el problema, idealizándolo demasiado para que pudiera ser tratable.

Básicamente la misma idea que se exponía al principio del capítulo al hacer referencia a las formas actuales de tratar los problemas de Codiseño.

Posteriormente, se produjo un cambio de orientación, y se adquirió una nueva metodología denominada *simbólica*. El simbolismo se caracterizó por la inclusión de tratamiento formal, asociado a técnicas de resolución algorítmicas. De esta manera, se dotó al sistema de un mayor grado de abstracción, al existir la posibilidad de proyectar mecanismos de deducción sobre el espacio de cálculo matemático. Es lo mismo que sucedió en la Informática cuando se dieron los primeros pasos hacia los lenguajes de programación estructurados, alejándose de los lenguajes máquina totalmente dependientes de la arquitectura tratada.

De esta manera, en el año 1981 y para conjugar las dos aproximaciones anteriores, Allan Newell, basándose en iniciativas similares previas, propuso un modelo de división del conocimiento conocido como *Teoría de los Niveles* [Newe81]. En este modelo, se crearon tres estados totalmente independientes que correspondían a los distintos grados de abstracción con que se podían abordar los problemas.

Primeramente, se tiene el nivel conexionista clásico, en el que los esquemas de solución corresponden a arquitecturas físicas que pretenden imitar los centros de razonamiento humano.

Como este nivel es demasiado bajo para enfrentarse a problemas de una cierta complejidad, se propone un nivel intermedio simbólico. En él se estudian las estructuras de datos y control, las cuales sustentan los algoritmos que simulan el comportamiento de las arquitecturas contenidas en el nivel inferior.

Por último, como estrato superior se adquiere el llamado nivel del conocimiento, en el que mediante modelado de tareas genéricas, se crea el algoritmo capaz de solucionar un cierto problema, que es absorbido por el nivel simbólico, y que a su vez es reflejo de la arquitectura conexionista.

Este nivel se rige por el llamado *principio de racionalidad*, concreción máxima del grado de abstracción aplicado al conocimiento.

Dados estos tres niveles, es posible, en teoría, alcanzar con cada uno de ellos la solución de cualquier problema propuesto, debiendo ser en todos los casos equivalente. Sin embargo, esto no quiere decir que encontrar la solución sea una tarea viable. Si se trata con elementos de demasiada entidad, puede ser que los niveles inferiores no dispongan de los recursos ni del tiempo necesarios para abordarlos. Por lo tanto, problemas situados más arriba en la escala conceptual se deben tratar con mecanismos de un nivel de abstracción más elevado.

Esto es precisamente una de las claves en la teoría presentada: *las técnicas asociadas a cada uno de estos estratos son independientes y no intercambiables*. Con otras palabras, las metodologías de un nivel no son utilizables en otro distinto, ya sea superior o inferior. Este concepto es el que se conoce como *cierre organizativo*. Gracias a esto se consigue una separación entre el conocimiento en sí, que es único, y su representación, que puede variar según el nivel al que se trate, lo mismo que un fenómeno físico puede adoptar distintos modelos matemáticos.

Otra de las claves de la teoría es la necesidad de aportar una llamada *inyección de conocimiento* al pasar de un nivel a otro superior. Es decir, a un supuesto observador local ligado a uno de los niveles, le resultaría imposible asumir los métodos de un nivel superior, porque para realizar el cambio de abstracción es necesario adquirir el conocimiento del nuevo entorno y su tipología asociada. Para ello, sería preciso proveer a dicho observador de las estructuras y métodos apropiados para simular el entorno de dicho nivel superior.

De los dos párrafos anteriores se puede establecer un nexo que nos lleve de vuelta al objeto de estudio original. La conclusión que me gustaría extraer se dividiría en dos puntos resumen:

- La necesidad de la existencia de varios niveles para abordar un problema de Codiseño, cada uno de ellos válido, pero con un grado de abstracción distinto.
- La obligatoriedad de dotar a cada uno de estos niveles de unas estructuras y métodos propios que reflejen las características intrínsecas de estas divisiones.

Por lo tanto, y según los comentarios previos, parece justificada la decisión de crear un nivel de abstracción superior al actualmente considerado, y que se adapte de una manera más natural a los cada vez más complejos problemas de Codiseño.

Es preciso dejar claro que no estoy afirmando la incorrección de los tratamientos actuales. Sin embargo, al alcanzar un cierto nivel de complejidad, pueden no ser factibles ni apropiados para los nuevos objetos de trabajo.

Una vez que he expuesto las justificaciones que me han llevado a considerar la necesidad de un grado de abstracción superior, es necesario definir el nuevo marco de trabajo, el *nivel de conocimiento* propuesto, y cómo se entiende dentro de las metodologías actuales existentes.

3.3 El punto de vista macroscópico.

Parece claro que como la principal motivación de esta idea es la reducción del tiempo de diseño, debido a la creciente complejidad de los problemas, sea precisamente en este punto donde el nuevo nivel de trabajo ha de hacer un mayor hincapié.

De esta manera, sería necesario dejar de utilizar la gran cantidad de datos que tradicionalmente se han generado en los procedimientos de planificación y asignación de hardware, y sustituirlos por otro tipo de información más manejable.

Una vez conseguido esto, es preciso crear nuevos métodos de trabajo que, a partir de los nuevos datos, sean capaces de llegar a una solución satisfactoria[2], en un tiempo de diseño mucho menor.

Aquí se ve la dualidad necesaria entre datos y métodos asociada a un nivel de trabajo. Realmente, sin la creación de ambos patrones se seguiría teniendo el mismo problema de diseño. Sin embargo, si se logra conjugar ambos, es posible llegar a un nuevo estadio de abstracción y generar nuevos esquemas de solución.

3.3.1. Características del punto de vista macroscópico.

Tras las discusiones previas, creo que las bases esenciales han quedado sentadas para la definición del nuevo nivel de trabajo, que explicaré en capítulos sucesivos. He denominado a este nuevo nivel *macroscópico*, por analogía con otros campos de la ciencia.

Su composición exacta, así como otro aspecto importante que es la relación o traducción con respecto a los datos y técnicas del nivel de abstracción inferior, se explicarán convenientemente en su momento.

Ahora, sin embargo, es preciso caracterizar no su contenido, sino su delimitación respecto al entorno, lo que equivaldría a su frontera imaginaria a través de la cual los datos no pueden fluir libremente. Este punto es muy importante en toda definición de marco de trabajo, pues explicita de una forma inequívoca en qué punto se está en cada momento, dependiendo del grado de abstracción de los datos que se quieran usar. Hay que recordar de

[2] Por satisfactoria se entendería aquella solución que aun perdiendo parte de calidad frente a la obtenida en un enfoque clásico, se consigue en un tiempo mucho menor. Obviamente, este compromiso de calidad y tiempo está sujeto a las necesidades propias de cada diseñador.

nuevo que en la teoría de niveles dichas divisiones son estancas, y en ningún caso se puede aplicar un concepto de delimitación borroso.

De esta manera, y de acuerdo con las afirmaciones hechas previamente, se delimita una aproximación a un problema de Codiseño dentro del nivel que he llamado macroscópico, si la menor unidad de información tratada es el nodo[3].

Quizás sea conveniente volver sobre esta definición, que aunque muy concreta, puede llevar a equivocación por una mala interpretación del concepto "menor unidad de información". Obviamente, no se puede ignorar la realidad de que en el interior de cada nodo, que a su vez representará una tarea escogida, existe una distribución de operaciones, que a la larga serán implementadas por un conjunto de unidades funcionales. Estas unidades darán lugar a un coste y un tiempo de ejecución de la tarea, que finalmente caracterizarán las propiedades del sistema codiseñado.

Sin embargo, el hecho de que ese nivel de operación exista, algo que está fuera de toda duda, no significa que tenga que ser considerado una vez iniciado el proceso de Codiseño.

En otras palabras, en un instante previo al comienzo de la metodología, esas operaciones pertenecientes a lo que a partir de ahora denominaré nivel *microscópico*, habrán de ser traducidas, mediante una inyección de conocimiento, para generar otras magnitudes equivalentes en el nivel macroscópico.

A partir de este punto, se comienza el proceso de Codiseño en dicho nivel, utilizando el conjunto de estas variables macroscópicas y sus técnicas asociadas, e ignorando los datos del nivel microscópico inferior. De hecho, ignorar no sería la palabra, porque ambos conjuntos de datos serían

[3] Como manifestación de tarea genérica una vez realizado el proceso de elección de la granularidad.

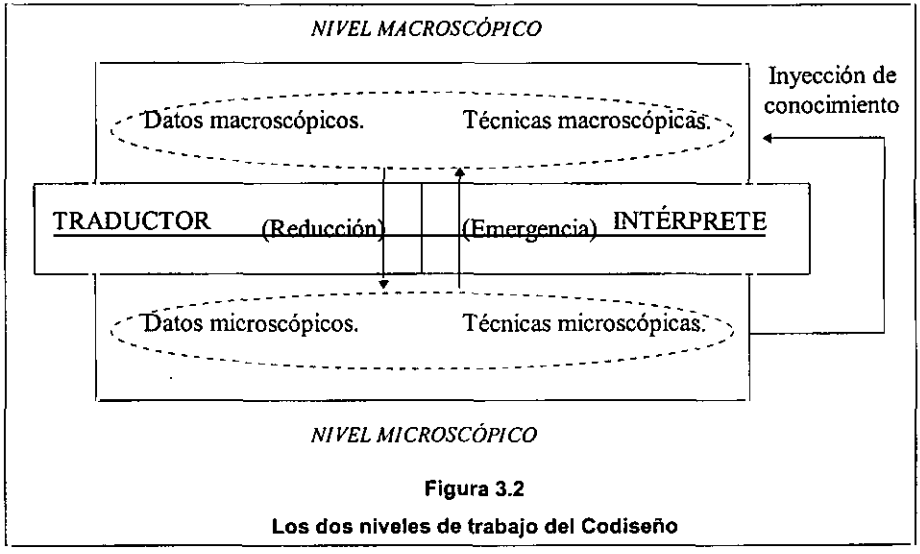


Figura 3.2

Los dos niveles de trabajo del Codiseño

equivalentes, sino que los datos microscópicos serían absorbidos y transformados en la frontera del nivel superior.

Es precisamente, esto que acabo de denominar como *inyección de conocimiento*, y que proviene de las teorías clásicas de niveles, lo que pretendo aportar con este trabajo: *la información extra que es necesario añadir para que el nuevo nivel de abstracción sea operativo y equivalente al actual.* En la Figura 3.2 se puede ver la comparación de estos dos niveles, junto con sus mecanismos de relación.

Recapitulando lo comentado en esta sección, se podría hacer un resumen de las distinciones que he introducido y con las que trabajaré a partir de ahora:

- El nivel macroscópico se caracteriza por el tratamiento exclusivo del nodo como pieza básica de información, relegando sus detalles internos a datos más genéricos. Estos datos han de tener la característica de ser manejables y sus técnicas han de estar dotadas de una gran rapidez de cálculo.

- El nivel microscópico, sin embargo, considera a la operación, asociada a un ciclo de reloj y a una unidad funcional, la mínima expresión de información, y basa todos los cálculos en técnicas que trabajen sobre ella.

Recuérdese que la introducción de este nivel macroscópico no es una decisión artificial, sino que surge de la imposibilidad clara de las técnicas clásicas para dar soluciones concretas a problemas de Codiseño en un tiempo razonable.

3.3.2. La necesidad de innovar frente a la oposición de la rutina.

Es normal que si algún diseñador de experiencia acostumbrado a tratar con problemas clásicos de la Síntesis de Alto Nivel ha leído hasta aquí, piense que estoy tratando el tema desde un punto de vista poco real. Quizás además opine que esa falta de nexo con la realidad permita dar lugar a diversas disquisiciones teóricas, pero que no pueden considerarse como un método para abordar problemas concretos, ni que las soluciones que puedan aportar se acerquen en precisión a las de las técnicas clásicas, aun con todos los problemas de tiempo que estas conllevan.

Esta primera aproximación es lógica, sobre todo si tradicionalmente se ha venido utilizando un punto de vista totalmente distinto. Sin embargo, aunque este cambio de orientación cueste trabajo es necesario realizarlo, si se ha optado previamente por abordar problemas de una determinada complejidad. Quizás la cuestión más costosa en este aspecto no sería la introducción de métodos novedosos, cosa que es habitual, sino que dichos métodos requieran un punto de vista y una forma de observar las cosas radicalmente diferente a las acostumbradas.

En esta sección trataré de dar una serie de justificaciones, no teóricas ni matemáticas, cosa que haré en el próximo capítulo, sino a un nivel práctico. De esta manera, considero que mediante el paralelismo con otros

campos de la ciencia, donde este tipo de cambios ya se han producido, se puede conseguir despejar las dudas de los más escépticos sobre la aplicación de este trabajo en un entorno real.

En casi todos los ámbitos científicos existen varios niveles de abstracción para tratar los problemas, y en la mayoría de los casos, estos niveles no fueron desarrollados simultáneamente.

Así, es normal que tras muchos años en los que la solución a un determinado problema era considerada de una cierta forma, se proponía una nueva aproximación, que si bien no contradecía en nada a la anterior, ofrecía la posibilidad de nuevos puntos de vista no explorados, que a su vez generaban ideas sobre nuevas aplicaciones de las teorías.

Por ejemplo se podría considerar el caso de la termodinámica clásica. El estudio del comportamiento de los cuerpos respecto al calor ha despertado siempre un gran interés. Desde un principio, todas las características relacionadas de estos cuerpos, y más concretamente de los gases, se han tratado desde un punto de vista macroscópico, circunstancia todavía adoptada en la actualidad.

Así, los parámetros de presión, volumen y temperatura representan conceptos asociados a la materia, pero que no tienen ninguna existencia en la realidad. Al contrario de lo que ocurre con otras magnitudes como la masa, que supone un valor intrínseco asociado a los elementos, los parámetros previos representan, a un alto grado de abstracción, el estado particular del conjunto de las innumerables partículas que forman el cuerpo.

De esta manera, la presión, volumen y temperatura de un gas no son sino la manifestación de la energía interna, cinética y probabilidad de choque de las moléculas individuales. Sin embargo, no es lógico usar estas magnitudes para caracterizar un gas cuando se trata de resolver un problema a los niveles normales en lo que se suelen plantear.

Con otras palabras, no es incorrecto calcular la energía interna de cada una de las moléculas de un gas, y extrapolar los resultados para hallar la temperatura global, simplemente es ineficiente, o imposible en la mayoría de los casos. Por eso, aunque está clara y presente la existencia de un nivel inferior microscópico, todo estudio se hace en función de parámetros más abstractos que son equivalentes a los propios de dicho nivel inferior.

Es curioso comprobar que en este caso, así como en la mayoría de los existentes en la ciencia, el paso de un nivel de abstracción a otro ocurrió justamente al contrario de lo que sucede en el Codiseño: de un nivel superior a otro inferior.

Este hecho se justifica debido a que en las disciplinas clásicas mencionadas se desconocía la existencia de la realidad a un nivel microscópico, y fue el descubrimiento de esta realidad lo que abrió las puertas a los nuevos parámetros y técnicas.

Sin embargo, en el caso que nos atañe no se ha producido este fenómeno de deducción, en el que a partir de teorías generales se ha llegado a conclusiones particulares, sino de inducción, ya que partiendo de disciplinas como la electrónica, se han ido aumentando los niveles de dificultad de los problemas hasta llegar a la Síntesis de Alto Nivel y al Codiseño.

Este hecho es lógico en todos los campos que han sufrido un incremento progresivo de los problemas que tratan, y en todos ellos se ha resuelto la situación de la misma manera: disminuir progresivamente el nivel de detalle hasta llegar al tratamiento de magnitudes más generales.

Dicho proceso es la base fundamental de la teoría estadística, que se utiliza en amplios campos de aplicación, desde los científicos hasta los sociales. Básicamente su principio es el mismo que el que estoy tratando de explicar, y se podría resumir en el hecho de que cuando el número de datos para manipular es suficientemente grande, se pueden realizar

consideraciones que son cumplidas sin tener que trabajar individualmente con cada uno de ellos. Dichas consideraciones no serían extrapolables a medida que los datos fueran disminuyendo, lo cual no significa que no sean consistentes, sólo que están asociadas a un cierto nivel de abstracción, y no tienen validez fuera de él.

Todavía es frecuente encontrarse con personas cuya disposición hacia la estadística es de desconfianza, bajo la sospecha de una falta de exactitud que redunda en la no precisión de los resultados. Sin embargo, la realidad es que funciona, y que si bien es indudable que se comete un cierto error frente al enfoque individualista de la información, este error tiende a ser muy aceptable si se compara con los órdenes de magnitud de los tiempos necesarios en el proceso de los datos.

Es preciso dejar claro que la teoría que da origen a este trabajo no está basada en la estadística, sino que aborda aspectos relacionados con el tratamiento generalista de la información y la no consideración de detalles individuales.

Con todo lo expuesto hasta aquí, parece justificado el hecho de proponer un nuevo nivel de abstracción dentro del Codiseño, por lo que en los próximos capítulos se explicará con detalle su composición y aplicación dentro del ámbito de trabajo actual.

Estimación de tiempo y coste en un entorno Macroscópico.

4

Tras haber expuesto todas las motivaciones que han inducido a proponer un cambio de visión en el modo de abordar los problemas de Codiseño, pasaré directamente a explicar las características de esta aproximación. Aunque en el capítulo siguiente se seguirán ofreciendo mejoras sucesivas, es en este donde reside el núcleo central del entorno ideado en su momento. Como posteriormente se comprobará, este modelo está plenamente justificado por los resultados que en él se han obtenido (§7). Se ha intentado que toda la formulación matemática sea lo más intuitiva posible, dentro de la rigurosidad exigida en estos casos, añadiendo ejemplos prácticos a los conceptos ofrecidos teóricamente.

4.1 Datos asociados al entorno: las variables macroscópicas.

Tras los razonamientos del capítulo anterior, debería haber quedado claro el concepto de hermetismo de un nivel de trabajo, o cierre organizativo, el cual ha de disponer de unos datos propios, relacionados de alguna manera con los nativos en los niveles adyacentes, y sobre los que trabajan unos métodos también particulares. Tanto datos como métodos son intransferibles, y por lo tanto no heredables de otros niveles.

Es en esta sección donde se van a explicar estos datos que caracterizan al nivel de trabajo [MaMM98] [MaMo98b], los cuales también heredan el calificativo de *macroscópicos*. Por supuesto que se podrían haber definido muchas más variables aparte de las básicas que se van a comentar, pero es importante mantener el denominado *principio de simplicidad*[1], al que me referiré con asiduidad de aquí en adelante.

No sólo es fundamental la definición inequívoca de las variables, sino también su relación respecto al nivel clásico microscópico. De esta manera, se puede contar con un mecanismo de traducción biunívoca en la frontera de ambos niveles, que permita la transformación sencilla de la información.

Así, se definen una serie de parámetros que caracterizan al nodo como unidad mínima, y que evitan usar información detallada de más bajo nivel. Estos parámetros se pueden dividir en dos grupos:

- Los intrínsecos a los nodos: Son datos asociados a cada nodo *per se*, es decir, debidos a sus propias características, y que permanecen constantes en todo momento del proceso de Codiseño. Son el *número de implementaciones*, el *tiempo* y el *coste* de cada una de estas implementaciones, y la *semejanza*.
- Los extrínsecos a los nodos: Son aquellos que definen alguna característica momentánea del nodo, y que van cambiando según evolucione el proceso de Codiseño. En este grupo se consideran el *estado* y el *solapamiento*.

[1] Principio que afirma que todo se ha de mantener lo más simple posible con tal de que se cumplan los objetivos propuestos, o de otra manera, que nada se debe complicar a no ser que sea estrictamente necesario para alcanzar las metas impuestas.

4.1.1. El número de implementaciones.

Es bien sabido que una especificación hardware puede implementarse de innumerables formas dependiendo del punto del espacio de diseño escogido. En otras palabras, un diseño hardware puede tener muy distintos pares de tiempo y de coste, según las restricciones que se hayan considerado.

Así, en el caso de querer minimizar la latencia, se optará por una implementación con alto grado de paralelismo, y por lo tanto, con una gran utilización de unidades funcionales y alto coste. Sin embargo, si se quiere optimizar el coste, la decisión irá encaminada hacia un diseño muy secuencial, en el que el reuso de las unidades hardware producirán un descenso en el coste final.

De esta manera, dada una especificación hardware, no es correcto hablar de su tiempo y coste asociados, sencillamente porque estos parámetros van ligados a la implementación y no a la especificación. Por lo tanto, dichos parámetros están indeterminados *a priori*, no pudiendo ser valorados hasta que se tome la decisión particular de cómo se va a estructurar el circuito.

No obstante, es normal encontrarse sistemas que consideran como parámetros "el tiempo y el coste" del nodo, como si ambos datos tuviesen un valor único e inconfundible.

Al hacer esto, se está fijando de antemano cómo va a implementarse la tarea en cuestión, lo cual limita en exceso la versatilidad del hardware, o dicho de otra manera el equilibrio entre el paralelismo y el reuso de sus unidades.

Un punto también claro es la imposibilidad de manejar todas y cada una de las implementaciones posibles de un nodo. Esto se debe a que su número puede ser muy elevado, y no todas ellas han de ser forzosamente

significativas. Por lo tanto, la consideración total de implementaciones, en el caso de ser posible, ralentizaría en demasía todo el proceso, sin ofrecer a cambio mejoras substanciales.

En consecuencia, el número de implementaciones que se deben considerar para cada nodo debe ser una característica macroscópica propia y determinada por el usuario. Así, en cada caso se podrá optar por un mayor o menor número de estas implementaciones, eligiendo las más adecuadas o significativas.

Es importante notar que aunque este número es limitado, recae en el propio diseñador la decisión de elegirlo, lo cual da una gran versatilidad al sistema. Dos implementaciones que *a priori* podrían ser significativas en la mayoría de los casos serían las de menor y mayor tiempo de ejecución, con un coste asociado inversamente proporcional.

De esta manera, se caracteriza el parámetro macroscópico *número de implementaciones*, *Z*, que se define matemáticamente:

$$Z_i = \{SW, HW_1, \ldots, HW_{m(i)}\},\ \forall i \mid 1 \le i \le Card(N) \qquad \textbf{[4.1]}$$

donde *N* es el conjunto total de nodos del sistema, y *m(i)* el número de implementaciones hardware escogidas para el nodo *i*. Notar que Z_i tiene un cardinal de *m(i)+1*, porque aparte de las implementaciones hardware, se tiene la implementación software única, la cual siempre ha de estar disponible para cualquier nodo.

4.1.2. El tiempo y el coste.

Quizás los parámetros de *tiempo* y *coste* sean los más comunes, por habituales, de los que conforman este modelo. Debido a las particulares restricciones que se manejan en la Síntesis de Alto Nivel, y por lo tanto en el Codiseño, donde se trata de buscar una solución con la mejor relación posible entre el tiempo y el coste, estos factores se han venido manejando desde el comienzo del diseño automático.

La herencia de ambos es obvia, si tenemos en cuenta que se ha definido un nodo como una tarea proveniente de la especificación inicial, el cual puede ser equivalente a un sistema completo de la Síntesis de Alto Nivel. Por lo tanto, los parámetros de tiempo y coste tradicionalmente calculados por esta disciplina, serían ahora los asociados al nodo de Codiseño correspondiente.

Según el párrafo anterior, calcular estos parámetros para cada nodo evaluado independientemente sería una tarea conceptualmente sencilla, que equivaldría a realizar un proceso de diseño clásico. Esto correspondería a efectuar la previamente definida *estimación de nodo* (§1.2.2.), anterior al comienzo del proceso de Codiseño.

Sin embargo, el hecho de estimar los mismos datos para varios nodos en conjunto, o dicho de otra maneara, realizar la *estimación de grafo* sobre la totalidad de tareas que finalmente va a conformar la partición hardware del sistema, es un proceso realmente complicado. Esto se debe a los efectos de compartición de hardware y posible paralelismo entre unidades inherente al sistema.

Esta complicación no se refiere a una dificultad teórica, sino a una complejidad en tiempo de cálculo contraproducente para los objetivos marcados. Como ya se comentó en el capítulo anterior, este proceso de estimación es uno de los cuellos de botella de todo el sistema, que es preciso acelerar.

En consecuencia, al disponer de una forma sencilla del tiempo y del coste de cada nodo por separado, es conveniente incluirlos dentro del conjunto de variables macroscópicas.

Como es obvio, estos parámetros no son específicos de un nodo, sino de su implementación en concreto. Por lo tanto, existirá un par de coste y tiempo asociado a cada pareja de nodo e implementación, ya sea hardware o software.

Dicho de otra manera, si se consideran todos los pares de coste y tiempo[2] para cada uno de los nodos de sistema, es posible establecer una relación clara y biunívoca entre este conjunto y el producto cartesiano de cada nodo con sus implementaciones válidas. De esta manera, definimos la relación:

$$P_i^j = (t_i^j, c_i^j), \quad \forall i \mid 1 \le i \le Card(N), \forall j \in Z_i \qquad \textbf{[4.2]}$$

Así, se obtienen un par de parámetros para cada posible relación de los distintos nodos con sus implementaciones diversas. Mediante la definición previa, es posible considerar ahora el conjunto de todos los parámetros asociados a un nodo dado, *i*, para cualquiera de sus posibles implementaciones:

$$P_i = \{P_i^{SW}, \ldots, P_i^j, \ldots, P_i^{HW_{m(i)}}\}, \quad \forall j \in Z_i \qquad \textbf{[4.3]}$$

Finalmente, si se consideran todos los nodos existentes en el diseño, se puede definir el conjunto total de parámetros de tiempo y coste, *P*:

$$P = \{P_1, \ldots, P_i, \ldots, P_n\}, \quad \forall i \mid 1 \le i \le Card(N), \; n = Card(N) \qquad \textbf{[4.4]}$$

El conjunto producto cartesiano representado por *P* contiene toda la información respecto a tiempos y costes de los nodos incluidos en la especificación. Es importante comprender la diferencia que supone considerar este conjunto de parámetros frente a la aproximación clásica en la que únicamente se disponía de una implementación prefijada para cada uno de los nodos.

[2] De entre las implementaciones escogidas por el diseñador.

4.1.3. La semejanza.

Quizás el parámetro que más se relaciona de una manera intrínseca con la aproximación macroscópica es el de la *semejanza*. Esto se debe a que dicho parámetro no era usado previamente, o no de la misma manera que aquí se considera.

Una de las primeras observaciones que se debería hacer respecto a la semejanza, es que es un parámetro binario[3], en contraposición de los explicados anteriormente, que eran unarios. Es precisamente esta relación entre pares de nodos la que le confiere un carácter especial.

Evidentemente, y aunque en la aproximación macroscópica no se desciende más abajo del nivel del nodo una vez comenzado el proceso de Codiseño, existe una información subyacente que no se puede ignorar. La composición de cada nodo es un factor clave para futuras consideraciones acerca de la posibilidad de compartir hardware, o la conveniencia de ejecutar dos tareas en paralelo.

Por lo tanto, esa relación *de facto* entre los nodos, que no surge de una forma artificial sino que existe de una manera natural, es preciso caracterizarla, labor esta realizada mediante el parámetro de semejanza.

Esta concepción un tanto sutil de que los nodos están relacionados a un alto nivel desde su generación no debe ser considerada una propuesta arbitraria. En casi todos los sistema basados en entes abstractos existen relaciones binarias entre los objetos, que acaban generando clases de equivalencia, como en los espacios vectoriales o la lógica de clases.

Siguiendo con las características de la semejanza, hay que decir que la única diferencia de la relación asociada a este parámetro con las de los

[3] *Binario* no significa aquí que comprenda valores de 0 y 1, sino que surge como relación entre dos nodos. Por contra, *unario* tiene el significado de estar ligado a cada nodo independientemente.

sistemas clásicos es que, debido a la naturaleza intrínseca de los objetos, dicha relación es borrosa. De esta manera, vendría representada como un índice real, entre 0 y 1, que indica cuán parecidos son dos nodos en términos de las unidades funcionales que los componen.

Dicho de otra manera, dos nodos que estén formados por unidades funcionales muy parecidas serán muy semejantes, y por lo tanto sería muy conveniente considerar su posible reuso para disminuir el coste final del sistema. Por otra parte, dos nodos con unidades funcionales muy dispares, podrían ser ejecutados en paralelo sin perjuicio del posible incremento de coste, ya que la posibilidad de reuso entre ellos sería muy baja.

Está claro que estas decisiones acerca del equilibrio entre paralelismo y reuso son críticas, y además numerosas, por lo que es conveniente abordarlas con un mecanismo rápido y preciso. Es exactamente este parámetro de semejanza el que ofrece esta posibilidad.

Otra cuestión también importante es que la semejanza no afecta sólo a un par de nodos determinados, sino que la implementación que se elija para estos dos nodos también es crítica.

Evidentemente, la implementación más lenta de un nodo contará con menos unidades funcionales que la más rápida, por lo que dependiendo de cuál de ellas se escoja, la comparación con otro nodo del sistema podrá variar considerablemente. De esta manera, existirá un valor de semejanza para cada par de nodos con cada una de sus posibles implementaciones.

La última precisión del parámetro semejanza es que es un factor relativo, y por lo tanto la relación que representa *no* es simétrica. Dicho de otra manera, la semejanza representaría cuán parecidos son dos nodos en proporción al hardware disponible del primero de ellos. Es obvio que el hardware común de dos nodos siempre será inferior al asociado a uno de ellos, con lo que la relación anterior siempre será menor que uno.

Así, definimos matemáticamente la semejanza entre los nodos *i1* e *i2*, con implementaciones *j1* y *j2* respectivamente como:

$$\kappa_{[i1,j1][i2,j2]} = \frac{HW \cap_{[i1,j1][i2,j2]}}{c_{i1}^{j1}} \qquad [4.5]$$

donde el numerador de la expresión corresponde a la intersección de las unidades funcionales de ambos nodos con sus implementaciones correspondientes. Aquí se contempla la propiedad mencionada anteriormente acerca de la no simetría de la semejanza, debido a que por lo general, los costes de ambos nodos son distintos:

$$\kappa_{[i1,j1][i2,j2]} \neq \kappa_{[i2,j2][i1,j1]} \qquad [4.6]$$

En este punto, se puede definir la matriz semejanza K, que estará formada por todos y cada uno de los posibles valores de las semejanzas individuales:

$$K = \{\kappa_{[i1,j1][i2,j2]}\}, \quad \forall i1, \forall i2 \mid 1 \leq i1, i2 \leq n; \forall j1 \in Z_{i1}; \forall j2 \in Z_{i2} \qquad [4.7]$$

Así, se obtiene cómo de semejantes son dos nodos, para cualquier par posible, con cualquier implementación escogida. Nótese aquí la importancia de que el parámetro semejanza esté constituido mediante una relación borrosa. En efecto, existen en la literatura aproximaciones que consideran la misma relación, pero de una manera clásica, es decir, sólo con posibles valores de 0 y 1. Esto equivaldría a considerar que si dos nodos son totalmente iguales (semejanza igual a 1) podrían compartir su hardware, que de hecho sería idéntico, y en caso contrario (semejanza igual a 0), no se podría compartir nada[4].

[4] Lo cual significa que si tenemos dos tareas que son prácticamente iguales, a excepción de una operación única en una de ellas, no se podría reusar nada de hardware, cuando en realidad el 99% del mismo sería coincidente.

Evidentemente, esta aproximación es muy poco realista, y prácticamente desaprovecha toda la capacidad de reuso inherente a la partición hardware.

4.1.4. El estado.

Hasta aquí se han comentado los parámetros macroscópicos intrínsecos a los nodos. En este punto y en el siguiente, se explicarán los extrínsecos, que van asociados a los nodos de una manera indirecta, y que cambian según avanza el proceso de Codiseño.

El primero de ellos se denomina *estado de un nodo*, y equivale a la implementación momentánea que tiene ese nodo en particular en un instante dado. Obviamente, se partirá de un estado inicial, que en nuestro caso se ha asociado al software, e irá cambiando según las decisiones que tome el particionador.

Así, un nodo puede comenzar asignado a la partición software, después ser asignado a la hardware, ir cambiando su implementación dentro de la misma partición, y regresar de nuevo al software.

Por lo tanto, la variación del estado de cada nodo es una función estocástica, no pudiendo ser predicha *a priori* mediante un análisis simple del problema.

Por supuesto, el rango de la función estado, S, para un cierto nodo ha de ser el conjunto de las implementaciones escogidas para dicho nodo. De esta manera:

$$S:\{i \mid i \in N\} \rightarrow \{j \mid j \in Z_i\} \qquad \textbf{[4.8]}$$

Realmente, la función estado representa cada una de las soluciones parciales que el sistema va adoptando. Si el proceso de Codiseño es detenido en un cierto punto y se observa la distribución de la función S, se obtendrá la mejor solución encontrada hasta ese momento.

De aquí se deduce que la solución final del sistema, aquella que se ha considerado como la mejor dentro del espacio de diseño explorado, corresponde al valor final de la función estado. Así, indicará cuál ha sido la composición de las particiones hardware y software, y con qué implementación se ha elegido cada uno de los nodos que han sido asignados a la parte hardware.

Dada una solución parcial, es preciso estimar su calidad para ser comparada con el resto de puntos de diseño. Para ello es preciso conocer los parámetros de tiempo y coste de cada uno de los nodos, de entre los posibles valores del conjunto *P*. De esa manera, se cuenta con la función de asignación de parámetros, *I*, la cuál dado un nodo con su estado actual, proporciona el par de tiempo y coste asociado:

$$I : \{i \mid i \in N\} \rightarrow P_i^{S(i)} \qquad \textbf{[4.9]}$$

En consecuencia, cada nodo, como entidad que puede instanciarse con distintas implementaciones en sucesivos instantes, tiene ligados de esta manera sus parámetros de tiempo y coste.

4.1.5. El solapamiento.

El último de los parámetros que conforman la estructura de datos del nivel macroscópico es el *solapamiento*. Al igual que ocurría con la función de estado, el solapamiento es un parámetro extrínseco de los nodos, que va cambiando a medida que avanza el proceso de Codiseño.

El solapamiento, como la semejanza, es un parámetro binario, ya que relaciona a pares de nodos. Esta relación también es borrosa, por lo que todo solapamiento vendrá asociado a un número real en el rango de 0 a 1.

Con este parámetro, se quiere caracterizar durante cuánto tiempo coincide la ejecución de dos nodos, para un instante determinado del proceso de Codiseño.

Así, el solapamiento tendría una relación directa con las latencias de ejecución de los nodos, al igual que la semejanza representaba las características comunes del coste hardware.

Está claro por lo visto en el punto anterior, que el estado al que se asigna un nodo es un factor muy importante que influye en la configuración de la solución final. Pero una vez que esta partición ha sido elegida, es igualmente relevante la posición temporal del nodo dentro de ella, o lo que es lo mismo, su situación respecto al resto de los nodos que la componen.

En el presente caso, esta consideración es trivial en la parte software, ya que al estar trabajando con un único procesador, el solapamiento de las tareas es nulo. Es decir, todas las tareas software se ejecutan de una manera secuencial. Pero en la partición hardware, por la propia naturaleza del mismo, existe el concepto intrínseco de paralelismo, el cual no se puede obviar si se quiere que la aproximación sea mínimamente real.

Así, dependiendo de la posición relativa de cada nodo, las características finales del hardware variarán considerablemente. Por regla general, y hablando en términos absolutos, una partición en la que exista un alto índice de solapamiento de los nodos (alto grado de paralelismo) será rápida y costosa, mientras que si el solapamiento es reducido (bajo grado de paralelismo), la partición será más lenta y más barata.

La definición matemática del solapamiento, σ, entre dos nodos *i y j*, vendría dada por:

$$\sigma_{i,j} = \frac{t \cap_{i,j}}{t_{i\,S(i)}} \qquad \textbf{[4.10]}$$

donde el numerador de la función representa al intervalo de tiempo en el que la ejecución de ambos nodos coincide. Es preciso hacer dos consideraciones respecto a esta definición:

La primera es que el solapamiento no depende de una manera *explícita* de las implementaciones de ambos nodos. Esto se debe a que dicho parámetro varía a medida que se avanza en el espacio de búsqueda, y dos pares idénticos de nodos con las mismas implementaciones, pueden tener solapamientos diferentes en distintos momentos.

La segunda se refiere a que, al igual que ocurría con la semejanza, el solapamiento es un factor relativo, dada la naturaleza del denominador de su definición, y por lo tanto, se cumple en general la siguiente desigualdad:

$$\sigma_{i,j} \neq \sigma_{j,i} \qquad \textbf{[4.11]}$$

Con las consideraciones previas, es posible definir la matriz global de solapamiento, Σ, de esta manera:

$$\Sigma = \{\sigma_{i,j}\} \ , \ \forall i, \forall j \mid 1 \leq i,j \leq n \qquad \textbf{[4.12]}$$

4.2 Planteamiento del problema: Objetivo del nivel macroscópico.

Una vez especificado el conjunto de variables macroscópicas que conforman el nivel de trabajo, y antes de pasar a describir los métodos que los caracterizan, es preciso definir claramente el objetivo que se intenta conseguir.

Evidentemente, con este sistema se ha tratado de dotar al proceso de Codiseño de una metodología eficaz que permita alcanzar el mejor resultado posible. De esta manera, la calidad de cada una de las soluciones parciales ha de poder ser comparada para optar por la mejor de ellas en cada uno de los casos.

Por lo tanto, el poder evaluar dicha solución parcial de la forma más precisa, constituye el mejor aval para finalizar el proceso satisfactoriamente. Al depender esta calidad de los parámetros de tiempo y coste del sistema,

su estimación correcta es condición necesaria para la consecución del objetivo propuesto.

Sin embargo, el proceso de estimación del sistema no es una tarea fácil, sino que constituye uno de los cuellos de botella del Codiseño. Aunque la obtención de estas estimaciones de tiempo y coste sobre las tareas individuales pueda ser una labor directa aplicando metodologías de Síntesis de Alto Nivel, el mismo proceso considerando el sistema completo aumenta considerablemente su complejidad, como se explicó en el capítulo anterior.

Por lo tanto, el cálculo de los parámetros de tiempo y coste de una solución intermedia constituirá el objetivo primordial de este nivel de trabajo, habiendo definido el conjunto de variables macroscópicas con el único fin de conseguirlo.

Es claro que esta estimación ha de contemplar unas características básicas que todo sistema orientado hacia problemas reales debe tener:

- Ha de ser capaz de contemplar tareas en paralelo, tanto aquellas que se encuentren en la parte hardware como entre dicha partición y la software.
- Tiene que considerar el reuso o compartición de hardware entre las distintas tareas que se asignen a esta partición.
- Debe poder hacer frente al hecho de que distintas soluciones parciales pueden constar de iguales nodos con diferentes implementaciones hardware.

Aunque los puntos previos podrían parecer obvios, o al menos bastante razonables, la mayoría de los sistemas no los consideran en su totalidad. Esto hace que buenos entornos de estimación y particionamiento pierdan parte de su incidencia al ser cotejados con problemas de campos reales.

Por eso se ha optado por que el sistema que aquí se propone adopte las características anteriores, que si bien no quiero manifestar que son las

más importantes[5], si son lo suficientemente significativas como para que su ausencia sea no deseada.

Dicho esto, y teniendo presente el primer apartado del capítulo, la formulación formal del objetivo previamente expuesto sería esta:

"Dada una solución parcial de un problema de Codiseño, generada por el particionador y representada por el grafo interno G={N, E}, la matriz de semejanza K, y un estado parcial de nodos, S, con la correspondiente asignación de parámetros I, calcular el tiempo de ejecución y el coste global asociados."

Obsérvese que esta definición encaja perfectamente con lo expuesto en párrafos anteriores, y que hace uso de las variables macroscópicas del nivel de trabajo.

Una anotación importante, que ya ha sido expuesta, es que la entrada al estimador es una solución parcial ya generada, y que por lo tanto las decisiones de qué nodos conforman cada partición y con qué implementaciones, ya han sido tomadas por el particionador. Por lo tanto, en la etapa de estimación se deben aceptar dichas decisiones en su totalidad, sin pretender su modificación.

El siguiente punto sería el referente al grafo de Codiseño G, y del que nada se ha dicho hasta el momento. Esta decisión ha venido dada por las circunstancias de que dicho grafo no forma parte realmente del conjunto de variables macroscópicas, sino que constituye una estructura interna sobre la que aplicar los métodos que se explicarán posteriormente. Realmente, el grafo es un entidad mutable, que en cada solución parcial adopta configuraciones distintas, y al considerarse un tipo de información superior al

[5] De hecho se podrían añadir muchas más consideraciones, como se verá en el capítulo acerca del trabajo futuro.

concepto de nodo, no es correcto tratarlo como un dato macroscópico en absoluto.

4.3 El grafo de Codiseño.

Todo sistema automático en cualquier campo de aplicación necesita, aparte de un soporte físico en el que ejecutarse, una estructura virtual que proporcione la interacción entre métodos y datos.

Esta estructura, denominada interna, tiene una gran importancia, ya que su adecuación al problema tratado hará que la mencionada interacción sea lo más fluida posible, y por lo tanto, más eficiente.

En otras palabras, una estructura no apropiada puede hacer que los métodos necesarios para procesar los datos adquieran una complejidad indeseada, o bien que estos datos no sean aprovechados de una forma óptima.

Por lo tanto, el sistema basado en la aproximación que aquí se presenta también necesita una estructura interna que lo soporte de una manera eficiente. Estructuras utilizadas en problemas de Codiseño hay muchas, cada una de ellas con sus ventajas e inconvenientes, dependiendo de las tareas que se quieran primar.

Sin embargo, básicamente todas ellas tienen la forma de grafo. Esto se debe a que dicha estructura es la más generalista y versátil dentro del campo informático, y por lo tanto, la más apropiada en la mayoría de los problemas, incluidos los de Codiseño.

Estos grafos pueden adoptar distintas formas más o menos complejas, para formar diferentes variantes en cuanto a su organización. Por ejemplo, es normal encontrarse propuestas de uso de redes de Petri [GrMa98] [RuMS96], que por sus características naturales, prestan una gran ayuda en problemas relacionados con la sincronización y en la representación jerárquica de niveles de datos.

Otras propuestas tienden al uso de los llamados hipergrafos [ErHB93] [EKPD98b], en los que un mismo conjunto de nodos comunes es unido mediante diversos grupos de aristas, formando varios grafos coexistentes. De esta manera es posible representar distintos tipos de información que fluyen entre las tareas de forma paralela, y operar sobre ellos convenientemente.

No obstante, todos ellos cumplen las propiedades morfológicas de los grafos, y si bien ciertas disposiciones ayudan a la realización de procesos concretos de una manera más eficiente, deberían ser capaces de soportar cualquier tipo de operación que se desee realizar.

En un principio, se planteó la duda de qué postura adoptar frente a la estructura interna: si utilizar un grafo especializado para facilitar la realización de los métodos macroscópicos, o bien un grafo más generalista, y por lo tanto más versátil, incluyendo sólo la información más necesaria.

Finalmente se optó por la segunda posibilidad, siguiendo uno de los principios que se ha tenido presente durante la realización de este trabajo: el principio de simplicidad. Si una de las propuestas que se han hecho es la concentración del esfuerzo en datos que realmente sean significativos en este nivel de trabajo, la consideración de información no esencial en la estructura interna rompe de alguna manera la tendencia inicialmente marcada.

Básicamente, y como todo grafo que sea estándar, la estructura interna elegida, que a partir de ahora denominaré grafo de Codiseño, se caracteriza por su conjunto de nodos y de aristas, como viene indicado en la definición del problema de la sección anterior, $G = \{N,E\}$.

4.3.1. El conjunto de nodos.

La característica básica que define a un grafo de Codiseño es su conjunto de nodos. De cómo son elegidos y de qué contengan, depende el

enfoque que se le quiera dar al problema, variando la granularidad en cada uno de los casos.

En la literatura se pueden observar distintas aproximaciones, desde grafos con una granularidad muy fina, en la que el nivel de detalle es importante, hasta estructuras de granularidad gruesa, mucho más generales.

El conjunto de nodos de la estructura presentada se podría catalogar dentro de este último grupo. Esto se debe a que al pretender abstraer el problema lo más posible, es necesario contemplar nodos con una cierta entidad, en los que los detalles puntuales se desvanezcan hasta relegar la información trascendente a aspectos más generales. Este hecho sólo se consigue encerrando tareas con un alto contenido de operaciones dentro de los nodos.

Por lo tanto, si la densidad de operaciones por nodo se procura que sea alta, el número total de estos tenderá a ser reducido. Sin embargo, es necesario tener presente que las etapas del sistema que aquí se presentan no tienen en consideración la elección de granularidad.

Esta decisión ha de ser tomada previamente con unas heurísticas adecuadas para que la división de la especificación inicial en nodos sea la apropiada. Por lo tanto, la elección del contenido de cada nodo no se discutirá aquí, sino que se considera un dato de entrada al problema.

En consecuencia, y a modo de resumen, los nodos del grafo de Codiseño serán aquellos que hayan sido producidos por la elección de la granularidad, con una tendencia a que no conformen un número excesivo.

Aparte de esto, la única característica especial que se le exige al sistema es que tenga un único nodo inicial (sin predecesores) y un único nodo final (sin sucesores). Esta elección se debe a facilitar la aplicación de las metodologías sobre la estructura, y no supone ningún tipo de limitación, ya que en el caso de que existan varios nodos sin predecesores o sin

sucesores, basta con introducir un nodo ficticio que no realice ninguna tarea y que pase a ser el inicial o final del sistema, según sea el caso.

4.3.2. El conjunto de aristas.

Si bien los nodos del grafo no han ofrecido ninguna dificultad dentro del ámbito conceptual, el conjunto de aristas sí que tiene unas características esenciales que lo diferencian de otras propuestas distintas.

La primera consideración sería el hecho de que este conjunto de aristas se ha escogido de tal forma que el grafo resultante siempre sea conexo y acíclico. Ambas condiciones son interesantes y se adaptan bien a las características de los problemas que se van a tratar.

La condición de grafo conexo reside en el hecho de que todas las tareas han de estar conectadas entre sí, ya sea por dependencias de datos (los datos de entrada de unas tareas son los resultados producidos por otras) o por dependencias de control (una tarea no puede comenzar hasta que acabe otra, debido a la unicidad de recursos con el procesador de la parte software). En caso contrario, en el que existirían dos subgrafos no conectados, se daría a entender que parte del sistema es totalmente autónomo respecto del resto. Esta consideración no es muy razonable dentro del marco de este trabajo, debido fundamentalmente al siguiente motivo.

Según se comentó en la estructura de la arquitectura escogida, los datos entre particiones se intercambian mediante el modelo de memoria compartida. De esta manera, todas y cada una de las tareas hardware han de estar conectadas, o bien a otra tarea hardware que genere sus datos de entrada, o directamente a una tarea software que le pase datos iniciales o generados.

Por otra parte, todas las tareas software han de estar conectadas, ya que al haber un único procesador, la ejecución ha de ser secuencial, y por lo

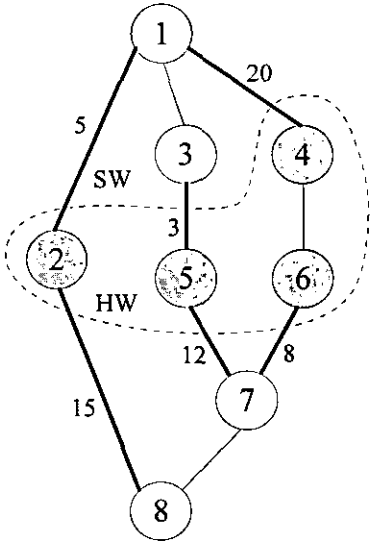


Figura 4.1
Grafo de dependencias de datos

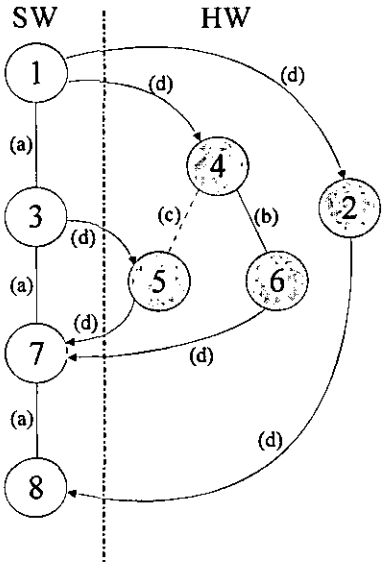


Figura 4.2
Grafo de Codiseño

tanto existe una dependencia de control. De los párrafos anteriores se concluye que todas las tareas deben estar relacionadas entre sí, o dicho de otra manera, que el grafo ha de ser conexo.

Para la consideración de grafo acíclico, se ha tenido en cuenta el conjunto de problemas a los que va destinada la presente aproximación. Estos problemas se podrían encuadrar dentro de las llamadas arquitecturas de flujo de datos, en los que dichos datos van atravesando distintas etapas de transformación hasta llegar al resultado final, sin que haya realimentaciones de ellos en ningún punto hacia tareas anteriores.

De esta manera, los bucles y demás estructuras repetitivas deberían estar contenidas dentro de los nodos de Codiseño, por lo que las dependencias de datos realimentadas no cruzarían nunca el umbral de la tarea.

La consideración de grafo acíclico que se ha hecho en esta aproximación no significa que sea imposible tratar problemas con realimentación de datos. Habría una complicación añadida en saber *a priori* el camino exacto que seguiría la información dentro de la arquitectura, para lo que se necesitarían técnicas de muestreo sobre cargas de datos características, pero la esencia de la metodología sería básicamente la misma.

Por lo tanto, es preciso dejar claro que la simplificación hecha acerca del flujo de datos del problema va dirigida hacia una mayor claridad en la explicación de la teoría macroscópica, y nunca en una limitación a la hora de abordar otros problemas igualmente tratables.

Realizadas estas consideraciones, es necesario delimitar ahora los distintos tipos de aristas introducidos para unir los nodos del grafo, lo que se ilustrará con un ejemplo. Primeramente, hay que realizar un estudio de las dependencias de datos, como se muestra en la Figura 4.1. La partición *software viene representada por los nodos blancos, y la hardware por los* nodos sombreados. Los dígitos que aparecen constituyen los tiempos de comunicación existentes entre los dos subsistemas.

Posteriormente, se crea el grafo de Codiseño, proveniente del grafo de dependencias ofrecido. Con estas premisas, el conjunto de aristas creado, que se refleja gráficamente en la Figura 4.2, viene catalogado por los siguientes tipos:

a) *Dependencias de datos y control software*: Estas aristas marcan el orden de ejecución de las tareas planificadas en software, que como se ha comentado anteriormente ha de ser secuencial debido a la consideración de que se tiene un único procesador. Por lo tanto, estas aristas formarán una cadena unidireccional de nodos a través de los cuales fluirá la información. Está claro que el orden de los nodos ha de respetar las dependencias de datos: las tareas se deberán ejecutar

después de la finalización de todas aquellas que producen algunos de sus operandos. Aparte de estos, puede existir un conjunto de nodos sin ningún tipo de dependencias entre sí, y por lo tanto intercambiables dentro de la cadena software. Todas las variaciones producidas de esta manera darían grafos válidos, pero es preciso determinar un orden claro e inequívoco para evitar ambigüedades dentro de la representación. Para ello, en el caso mencionado anteriormente, se fija el orden dado por el compilador del sistema[6], creando una dependencia de control entre los nodos implicados que les fuerza a una ejecución secuencial.

b) *Dependencias de datos hardware*: Las aristas que conforman este conjunto delimitan claramente la estructura jerárquica de las tareas hardware respecto a sus datos. Estas aristas no introducen ninguna dificultad conceptual, ya que se derivan directamente de la propia especificación inicial del problema. De esta manera, tareas que se ejecutan de una forma lineal en software, pasan a tener una dependencia arbórea al ser movidas a hardware, estructura que es producida por el conjunto de dependencias de datos. Así, nodos que tienen algún tipo de dependencia han de ser ejecutados de una manera secuencial, mientras que aquellos que no la tienen pueden ser ejecutados en paralelo, aprovechando esta característica intrínseca al hardware.

c) *Dependencias de control forzadas en hardware*: Está claro que el conjunto de aristas anteriores delimita qué tareas hardware pueden ejecutarse en paralelo, debido a la ausencia de dependencias. Sin embargo, el hecho de que dos nodos puedan ejecutarse en paralelo no

[6] Dada la especificación inicial en la que todas las tareas están en software, el compilador del sistema ha de generar una ordenación en ellas, que es la que sigue el procesador para ejecutarlas. Por supuesto este orden ha de cumplir las dependencias de datos.

significa que deban hacerlo, sólo que existe esa posibilidad. Así, se puede elegir bajo ciertas circunstancias tratar de minimizar el coste del sistema, y por lo tanto incrementar al máximo las posibilidades de reuso. De esta manera, lo que conviene es reducir el grado de paralelismo del sistema, aun en las tareas que intrínsecamente son independientes. Para expresar el significado de esta idea, he introducido este conjunto de aristas forzadas, denominado formalmente E_f. Así, dichas aristas introducen dependencias que no existen, o dicho de otra manera, que son ficticias, pero bajo ciertos requerimientos indican al sistema la necesidad de obtener una secuencialidad forzada. Estos enlaces, que como toda la solución parcial son generados por el particionador, constituyen un elemento para explorar eficientemente el equilibrio entre paralelismo y reuso hardware del sistema, como se explicará en su momento.

d) *Dependencias de datos entre el hardware y el software*: Es obvio que entre las dos particiones del sistema existe un paralelismo natural que siempre conviene explotar. No hay ninguna ventaja en la ejecución secuencial, como ocurría dentro de la partición hardware, por lo tanto, la única limitación o dependencia entre estos dos subsistemas es la debida estrictamente a los datos. Dicho de otra manera, las tareas en hardware y software han de transmitirse información, para que los resultados de unas de ellas pasen a ser los operandos de otras. Este intercambio de datos se realiza, como se explicó previamente, mediante el modelo de memoria compartida a través del bus del sistema. Una característica fundamental de esta comunicación de datos es el alto tiempo que se consume en su realización, por lo que constituye uno de los cuellos de botella del sistema. Como se verá posteriormente, una de las tareas claves del particionamiento reside en la distribución adecuada

de tareas, de tal forma que el tiempo de comunicación se reduzca al máximo.

Tras la explicación del grafo de Codiseño, formado por sus conjuntos de nodos y aristas, la estructura de trabajo ha quedado plenamente fijada. Ahora, es el momento de pasar a explicar los métodos macroscópicos, que junto con los datos expuestos en la primera sección de este capítulo, van encaminados a estimar el tiempo de ejecución y el coste del sistema.

4.4 Estimación del tiempo de ejecución del sistema.

La primera referencia que se va a calcular sobre la solución parcial generada por el particionador es el tiempo de ejecución. Para hallar este parámetro, es preciso considerar la posibilidad de paralelismo entre las tareas, lo que sin duda reportará un gran beneficio respecto del caso trivial de ejecución secuencial.

Realmente, observando el grafo de Codiseño de la sección anterior, el cálculo del tiempo de ejecución se simplifica enormemente. Para ello, y antes de entrar en consideraciones posteriores, hay que hacer un análisis sobre dicho grafo y estudiar si efectivamente comprende toda la información necesaria.

4.4.1. Completitud de información del grafo.

Básicamente, el grafo de Codiseño debería contener toda la información influyente en el tiempo final de ejecución. Esta afirmación se refiere a que todo aquello que conlleve tiempo, y que tenga relación con la generación del resultado final, debería ser tenido en cuenta.

Así, se ve que todo flujo de información asociado al problema está representado en el grafo ofrecido. Por una parte, las tareas software se comunican la información a través de las aristas de tipo (a). Del mismo modo, las tareas hardware hacen lo propio mediante las aristas de tipo (b).

El intercambio híbrido de datos (de software a hardware o viceversa) se produce a través de la comunicación entre particiones representada por las aristas de tipo (d). Estas aristas, igualmente transmiten un valor inicial desde la memoria del sistema a una tarea hardware, o en sentido contrario si se trata de un resultado producido por dicha partición.

Por otra parte, el mecanismo de secuenciación de tareas, o flujo de control, también está contenido dentro del grafo. El permiso de ejecución de las tareas software fluye por las aristas de tipo (a), y la serialización de tareas de hardware no dependientes de los datos se realiza mediante el conjunto de aristas (c).

Se ve que ambos flujos, de datos y de control, según definición previa, están correctamente representados dentro de la estructura interna. Además, no existe ningún otro tipo de información dentro del sistema: no se requiere nada más para conocer los operandos de las tareas y se sabe cuándo han de ejecutarse. Todo conocimiento que se añada sobre esto es redundante, a efectos del problema que se está tratando, cuyo fin es alcanzar el objetivo propuesto del cálculo del tiempo de ejecución.

A partir de esta exposición, propongo como definición de tiempo de ejecución de un sistema aquel lapso de tiempo en el que se encuentra activo cualquier subconjunto de los flujos de información, ya sea de datos o de control.

Por lo tanto, y teniendo en cuenta que toda esta información se halla presente en el grafo, el tiempo que tarda en trasladarse desde el nodo inicial hasta el final, dará de una manera inequívoca el tiempo de ejecución del sistema.

Repasando el párrafo anterior, se llega a la conclusión de que el tiempo así explicitado se corresponde biunívocamente con el tiempo asociado al camino crítico del sistema.

De esta manera, y gracias a la definición estructural del grafo de Codiseño, el cálculo del tiempo de ejecución se reduce a hallar el camino crítico de la estructura, tarea esta de una baja complejidad y para la que existen numerosos algoritmos ampliamente probados. En el siguiente punto comentaré cuál ha sido la técnica elegida, y cómo se relaciona con los datos macroscópicos disponibles.

4.4.2. Cálculo del camino crítico.

El primer paso para el cálculo del camino crítico consiste en la definición de *orden topológico*. El orden topológico (o.t.) de un grafo es una lista ordenada de nodos, en la que todas las clases de dependencias son respetadas. Por lo tanto, todos los predecesores de un nodo dado aparecerán antes que él en la lista, y todos sus sucesores aparecerán después.

Está claro que el orden topológico de un grafo no es único, debido a que todo subconjunto de nodos sin ningún tipo de dependencia entre ellos admite múltiples ordenaciones. En concreto, cualquier permutación dentro de estos subconjuntos generaría un orden topológico válido del grafo.

Una de estas ordenaciones válidas, de entre todas las posibles, sería la ofrecida por el compilador del sistema al planificar todas las tareas en software, lo que equivaldría a las dependencias de datos y control software contenidas en el grafo. Sea como fuere, es preciso calcular cualquier orden topológico correcto para proceder a la estimación del tiempo de ejecución del sistema.

A partir de este momento, se definen dos instantes de tiempo para cada nodo, aquellos en los que más pronto (SS) y más tarde (LS) puede comenzar su ejecución. Estos dos tiempos delimitan un rango de movilidad para el nodo de tal manera que no se incremente el tiempo global del sistema.

Estos dos puntos temporales pueden recordar a los clásicos *ASAP* y *ALAP* de la Síntesis de Alto Nivel, los cuales participaban de igual manera en el cálculo del tiempo de ejecución de un circuito. Sin embargo, el motivo de denominarlos de otra forma se debe al hecho de que ahora se están considerando tiempos macroscópicos asociados a los nodos, y no tiempos de unidades funcionales internas.

De esta manera, se marca una diferencia clara entre los dos niveles de trabajo para evitar cualquier tipo de ambigüedad.

Para calcular ambos tiempos de cada nodo, SS y LS, es preciso hallar el máximo y mínimo tiempo en el que la información puede fluir a través de ellos. El algoritmo propuesto para obtener el conjunto de valores SS viene dado por:

1.- Asignar al primer nodo en o.t., α, el tiempo 0:

$$SS(\alpha) := 0; \qquad \textbf{[4.13]}$$

2.- Para cada nodo *i*, en o.t., calcular *SS(i)* de la siguiente forma:

$$SS(i) = \max_{j \in Pred(i)} \{SS(j) + t_j^{S(j)} + \beta \cdot comm(j,i)\}$$
$$\beta = \begin{cases} 1 & \text{si } i \text{ y } j \text{ en diferentes particiones} \\ 0 & \text{en caso contrario} \end{cases} \qquad \textbf{[4.14]}$$

El valor de *comm(j,i)* representa el tiempo necesario para transferir datos desde el nodo *j* hasta el nodo *i*, usando el bus del sistema. Dicho valor sería el asociado a las aristas del grafo que contienen las dependencias de datos entre el software y el hardware. Este tiempo sólo está activo cuando ambos nodos estén asignados a diferentes particiones.

Por la expresión anterior, se deduce que lo más pronto que puede comenzar una tarea es cuando recibe toda la información necesaria de sus predecesores. La condición de totalidad viene garantizada por la operación máximo.

S(1) = SW	I(1) = (100, 0)
S(2) = HW_2	I(2) = (40, 15)
S(3) = SW	I(3) = (45, 0)
S(4) = HW_1	I(4) = (25, 65)
S(5) = HW_3	I(5) = (10, 130)
S(6) = HW_1	I(6) = (12, 95)
S(7) = SW	I(7) = (75, 0)
S(8) = SW	I(8) = (90, 0)

Figura 4.3

Valores de las funciones asignadas

Ahora, es preciso calcular el conjunto de tiempos LS, de manera análoga al algoritmo presentado. Primero, se considera el último nodo en o.t., ω:

3.- El valor de LS que se define para ω es:

$$LS(\omega) := SS(\omega) \qquad \textbf{[4.15]}$$

4.- Después, para cada nodo *i*, en o.t. inverso, hallar *LS(i)* de esta manera:

$$LS(i) = \min_{j \in Suc(i)} \{LS(j) - t_i^{S(i)} - \beta \cdot comm(i,j)\} \qquad \textbf{[4.16]}$$

El parámetro β es el mismo definido para calcular LS. De esta manera, lo más tarde que puede comenzar una tarea, sin perjuicio de retrasar la ejecución global del sistema, es el equivalente al mínimo tiempo de comienzo exigido por sus sucesores, menos el tiempo que lleva en realizarse la tarea en cuestión.

Así, al finalizar el algoritmo previo, se obtendrán distintos pares de tiempos (SS,LS) para cada nodo del sistema. El objetivo de todo este desarrollo es encontrar el tiempo asociado al camino crítico del grafo,

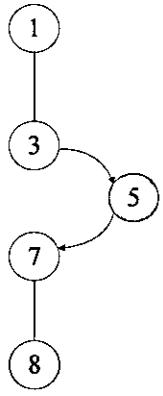


Figura 4.4
Camino crítico

SS	
1:	**0**
2: 0+100+5=	**105**
3: 0+100=	**100**
4: 0+100+20=	**120**
5: max(100+45+3, 120+25)=	**148**
6: 120+25=	**145**
7: max(100+45, 148+10+12, 145+12+8)=	**170**
8: max(170+75, 105+40+15)=	**245**

Figura 4.5
Cálculo de *SS*

LS	
8:	**245**
7: 245-75=	**170**
6: 170-12-8=	**150**
5: 170-10-12=	**148**
4: min(148-25, 150-25)=	**123**
3: min(148-3-45, 170-45)=	**100**
2: 245-15-40=	**190**
1: min(100-100, 123-20-100, 190-5-100)=	**0**

Figura 4.6
Cálculo de *LS*

equivalente al tiempo global de ejecución del sistema. Para identificar los nodos de este camino crítico, basta con encontrar aquellos que cumplan la siguiente condición:

$$SS(i) = LS(i) \qquad \textbf{[4.17]}$$

Estos nodos se caracterizan por tener un rango de movilidad nulo, o dicho de otra manera, por tener prefijado su instante exacto de comienzo, que es precisamente la definición del camino crítico.

Por lo tanto, y teniendo en cuenta que el nodo final de sistema, ω, es único, y que siempre pertenece al camino crítico, se tiene que el tiempo global de ejecución del sistema, T, viene dado por:

$$T = SS(\omega) + t_{\omega}^{S(\omega)} \qquad \textbf{[4.18]}$$

Tras la explicación de este algoritmo, se pretende su aplicación al ejemplo propuesto en la sección anterior. Para ello, se consideran las funciones de estado, S, y de asignación de parámetros, I, que se muestran en la Figura 4.3.

Los resultados del cálculo para los tiempos *SS* aparecen en la Figura 4.5, y los de los tiempos *LS* en la Figura 4.6. Se comprueba fácilmente que el camino crítico está formado por los nodos 1, 3, 5, 7 y 8 (Figura 4.4).

Como se puede observar, la complejidad de este proceso es realmente baja. Se necesitan simplemente dos recorridos de los nodos en orden topológico para llevarlo a cabo. Para el cálculo de los tiempos de cada nodo, es preciso consultar la lista de sus sucesores y predecesores, pero como la conectividad[7] del grafo es normalmente menor que el número total de nodos del sistema, se tiene que este factor suele ser despreciable respecto al primero, y por lo tanto la complejidad del algoritmo vendría acotada por O(n).

Está claro por este algoritmo que aquellos nodos pertenecientes al camino crítico tienen un tiempo de ejecución fijo.

Sin embargo, para todos los nodos que se encuentren fuera de él, sí es posible elegir en un rango de posibilidades cuál debería ser su tiempo de comienzo. En otras palabras, los nodos fuera del camino crítico disponen de un rango de movilidad. Este segmento temporal en el que el nodo puede oscilar, viene limitado por su par de tiempos (SS,LS) calculado anteriormente. Cuanto mayor sea la diferencia entre ellos, más grande será la movilidad del nodo, y por lo tanto más libertad se tendrá a la hora de decidir su tiempo de ejecución.

De esta manera, para cada nodo *j* en estas circunstancias, se puede definir un intervalo de tiempos, I_1, en el que su ejecución debe comenzar forzosamente:

$$I_1 = [SS(j), LS(j)] \qquad \textbf{[4.19]}$$

Igualmente, el tiempo de finalización del nodo también debe estar acotado, ya que su duración es finita y conocida. Por lo tanto, es posible

[7] Número medio de aristas por nodo.

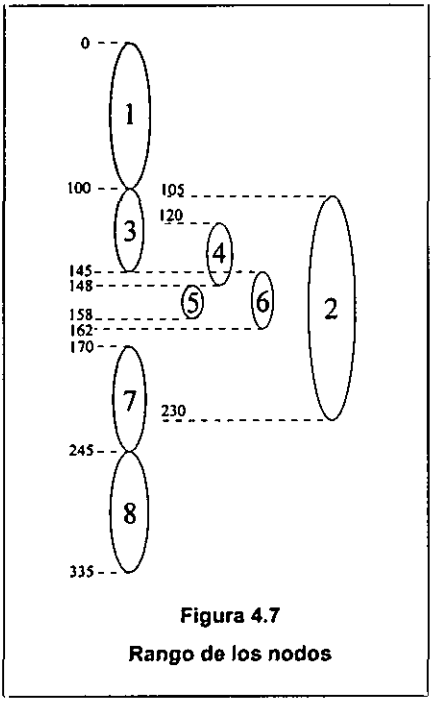


Figura 4.7

Rango de los nodos

definir un segundo intervalo para cada uno de estos nodos, I_2, en el que su ejecución ha de finalizar forzosamente:

$$I_2 = [SS(j)+t_j^{S(j)},\ LS(j)+t_j^{S(j)}] \qquad \textbf{[4.20]}$$

Estos dos intervalos de movilidad crean un espacio temporal en el que cada uno de los nodos del sistema puede estar activo. De esta manera, se define el rango de un nodo *j*, *rg(j)*, como el intervalo dado por:

$$rg(j) = I_1 \cup I_2 = [SS(j),\ LS(j)+t_j^{S(j)}] \qquad \textbf{[4.21]}$$

Esta definición es válida para cualquier nodo, incluidos aquellos dentro del camino crítico. En la Figura 4.7 se pueden observar los rangos obtenidos para el ejemplo tratado en este capítulo. El concepto de nodo activo se corresponde con la posibilidad de que dicho nodo se esté ejecutando en un

instante de tiempo dado. Este punto merece una consideración especial, ya que es este hecho el que determinará posteriormente la capacidad de un par de nodos para compartir sus unidades funcionales. En efecto, si la actividad de dos nodos se produce simultáneamente, la capacidad de reuso decae drásticamente, debido a la necesidad conjunta de uso del hardware

Es precisamente la asignación de un tiempo exacto de comienzo $t_s(i) \in I_1$, para cada nodo i, lo que hará que este factor de reuso varíe, por lo que la realización de esta tarea, que se denomina *planificación macroscópica*, es el siguiente punto por tratar dentro del proceso de estimación.

4.5 La planificación macroscópica.

Como se ha comentado anteriormente, la asignación de un tiempo de comienzo exacto a cada nodo fuera del camino crítico es una labor imprescindible antes de proceder a la estimación del coste del sistema.

Esta tarea, obviamente, recuerda a los procesos de planificación de Síntesis de Alto Nivel, en los que cada una de las operaciones debían asignarse a una etapa de control concreta. Sin embargo, se ha utilizado aquí de nuevo el término macroscópico, para dejar bien claro que en estos momentos nos referimos a la planificación de los nodos globales dentro del grafo de Codiseño.

Por supuesto que esta decisión marcará a las unidades funcionales internas de dichos nodos, que tendrán que ser planificadas en algún instante entre el comienzo y el final de la ejecución de la tarea a la que pertenezcan.

Está claro por todas las consideraciones previas, que lo que más interesa una vez conocido el tiempo de ejecución del sistema es obtener un coste global lo más bajo posible. Por lo tanto, la planificación debería ir encaminada hacia ese objetivo. De esta manera, es necesario conjugar

adecuadamente las variables macroscópicas disponibles con el fin de conseguir este propósito.

Primeramente, y según lo explicado al comienzo de este capítulo, es claro que la semejanza de un par de nodos influye determinantemente en la capacidad de reuso. Así, convendrá que aquellos nodos altamente semejantes puedan compartir sus unidades funcionales siempre que sea preciso. Esto reportará sin duda un gran ahorro en coste, ya que por la definición de semejanza, las unidades que se podrán beneficiar de esta política serán muchas.

Si bien la matriz de semejanza K tiene una clara repercusión en este estudio, no es menos cierto que igual relevancia tiene Σ, que representaba la matriz de solapamiento.

El solapamiento era considerado un parámetro macroscópico extrínseco, y aquí se ve claramente el porqué de esa definición. Dependiendo de qué planificación se escoja, este factor irá variando para cada par de nodos.

Evidentemente, si la semejanza se consideraba un factor beneficioso *para la capacidad de reuso, con el solapamiento se tiene totalmente lo* contrario. Nodos idénticos, y por lo tanto con una semejanza máxima, no podrían compartir nada de su hardware en el caso de que la planificación escogida los forzase a una ejecución paralela.

Así, estos dos son los factores necesarios para realizar una correcta planificación macroscópica. El punto clave reside en cómo conjugarlos para expresar adecuadamente la semántica introducida en los párrafos previos: minimizar el solapamiento de los pares de nodos con semejanza máxima.

Es esta semántica la que constituye el punto clave en el proceso de planificación, que no es otro sino la elección de la función objetivo que se pretende minimizar. Por lo tanto, y a la vista de las explicaciones dadas hasta ahora, una función lógica sería la siguiente:

$$\sum_{i \in HW} \sum_{j \in HW} (\sigma_{i,j} \cdot \kappa_{[i,S(i)],[j,S(j)]}) \qquad \textbf{[4.22]}$$

Así, se desea que el producto de la semejanza y el solapamiento se mantenga mínimo. Para conseguir esto, en el caso de un par de nodos con una semejanza alta, no hay más remedio que disminuir drásticamente su solapamiento, que es precisamente el objetivo marcado al principio.

Cabe hacer un comentario añadido acerca de esta función. La aparición explícita de los sumatorios extiende la aplicación del objetivo a todos y cada uno de los pares de nodos hardware del sistema. Obviamente, los nodos software no interesan en este punto, ya que no contribuyen en manera alguna al reuso de unidades funcionales.

Nótese además que la presencia de los dos sumatorios hace considerar todos y cada uno de los términos recíprocos[8]. Esto es necesario, ya que tanto la semejanza como el solapamiento no son magnitudes simétricas, y el hecho de invertir el orden de sus índices hace variar el valor de la expresión.

Otra consideración importante es la utilización del concepto de solapamiento en este contexto. Según la definición ofrecida por [4.10], el solapamiento consistía en la intersección del tiempo de ejecución de dos nodos, o dicho de otra manera, su segmento temporal coincidente.

Sin embargo, en este punto, todavía se desconoce el momento exacto de comienzo y finalización de cada nodo, por lo que esta definición no se puede aplicar directamente. Para solventar este problema, se utiliza el concepto de *rango* introducido en la sección anterior. De esta manera, no se considera el tiempo en el que el nodo se está ejecutando, que es desconocido, sino el segmento en el que puede estar activo. Así, el

[8] Término recíproco de uno dado es aquel que se obtiene invirtiendo de orden todos los subíndices de la semejanza y el solapamiento: $[i,j] \rightarrow [j,i]$.

concepto extendido de solapamiento que se utiliza en este punto viene dado por:

$$\sigma_{ij} = \frac{|rg(i) \cap rg(j)|}{|rg(i)|} \qquad \textbf{[4.23]}$$

Mediante las barras verticales se ha representado el tamaño del segmento temporal encerrado entre ellas. Se ve que esta definición de solapamiento extiende perfectamente la original, y que ambas coinciden en el caso de que los nodos ya dispongan de un tiempo de comienzo concreto[9].

El siguiente punto consiste en realizar la elección del algoritmo de planificación que se va a utilizar. Evidentemente, existen múltiples métodos que realizan esta función, todos ellos ampliamente probados.

En este caso, lo más interesante es la utilización de uno que no conlleve excesivo tiempo, lo que va en concordancia con la motivación de este trabajo. Por eso, se propone uno basado en la bien conocida metodología de *planificación por listas* [EKPD98a].

El fundamento de esta técnica reside en ir planificando los nodos en un orden calculado, mediante una función de prioridad previamente seleccionada aplicada a una lista.

La complejidad de los algoritmos basados en listas se puede acotar mediante $O(n^2 \cdot log\ n)$, lo que no supone una inversión excesiva de tiempo, comparada con otras aproximaciones existentes.

Los pasos del algoritmo propuesto se detallan a continuación.

1.- Planificar todos los nodos que se encuentren en el camino crítico. Esta es una operación trivial, ya que para este conjunto de nodos la

[9] Ya que en ese caso $rg(i) \equiv t_i^{S(i)}$, y se comprueba fácilmente que ambas definiciones coinciden.

movilidad existente es nula, y por lo tanto, cada uno de los tiempos de comienzo vienen ya prefijados.

Planificar todos los nodos software en su *SS*. Como estos nodos no tienen ninguna influencia sobre futuras consideraciones acerca del reuso hardware, su planificación resulta trivial en este sentido. Se ha elegido el *SS* por un criterio de prontitud en las tareas, que siempre resulta interesante por si se produce algún retraso en la ejecución debido a razones ajenas al propio sistema.

2.- Seleccionar el nodo hardware *i* que tenga un mayor factor de conflicto χ, respecto de otro nodo hardware, *j*, ya planificado. El factor de conflicto viene dado por:

$$\chi_{i,j} = \frac{(\sigma_{i,j} \cdot \kappa_{[i,S(i)][j,S(j)]} + \sigma_{j,i} \cdot \kappa_{[j,S(j)][i,S(i)]})}{rg(i)} \qquad \textbf{[4.24]}$$

Este factor es el referido anteriormente como la función de prioridad. Se ve que mediante su uso, los pares con mayor semejanza y con un posible mayor solapamiento son planificados primero, para minimizar este último factor de inmediato, y por lo tanto, cumplir así con los objetivos propuestos. También es trascendente el denominador de la función, el cual prima la elección de nodos con un bajo rango de movilidad. Este hecho es importante, ya que a medida que avanza la planificación, los rangos de los nodos tienden a disminuir[10], y es preferible tratar primero los menos móviles para tener más capacidad de decisión.

En el caso de que ningún nodo hardware perteneciera al camino crítico, la condición de que *j* esté ya planificado no se cumpliría en el primer paso del

[10] Esto es así porque al planificar todo nodo, sus antecesores y sucesores se ven limitados por la elección hecha.

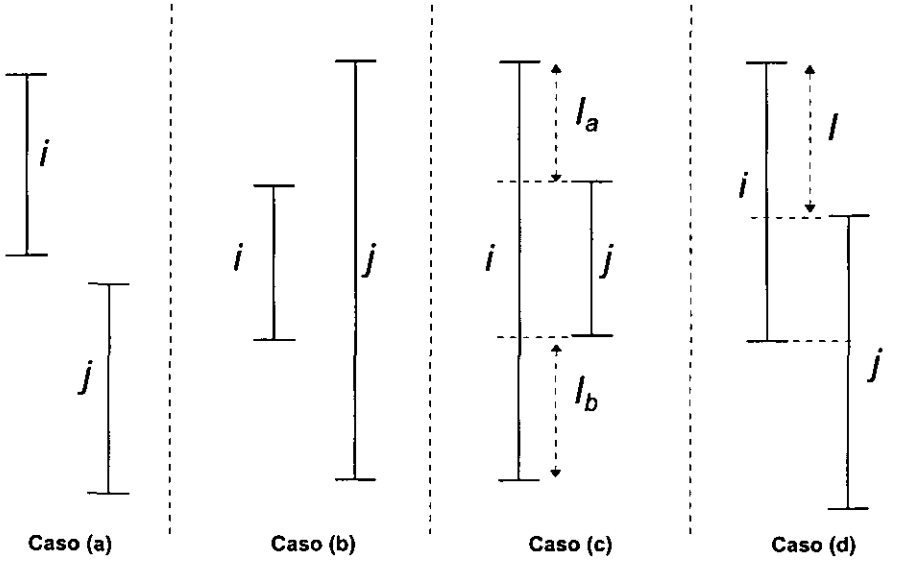


Figura 4.8
Distintas posiciones relativas de los intervalos

algoritmo. De ser así, escoger *i* y *j* como el par de nodos cualesquiera con mayor factor χ.

3.- En este punto, es preciso estudiar la posición relativa de *i* y *j*, mediante comparación de sus rangos de movilidad. Se trata de reducir al máximo su nivel de paralelismo, dentro de lo posible. Para ello, se calculan dos intervalos asociados al nodo *i*, denominados l_a, l_b, y que representan las *áreas del rango original de i, rg(i), que no coinciden temporalmente con el* rango de *j*. De esta manera se cumple que:

$$l_a \cup l_b = rg(i) - [rg(i) \cap rg(j)] \qquad \textbf{[4.25]}$$

Aquí, el operador *menos* (-) representa el cálculo del subconjunto del primer operando que a su vez no es subconjunto del segundo operando. Hay que decir que no en todos los casos deben existen los dos intervalos definidos,

pudiendo obtener solamente uno de ellos, o incluso ninguno, como se observa en los casos (a) y (b) de la Figura 4.8.

El propósito de este punto consiste en estimar en cuál de estos intervalos se debería planificar el nodo *i*, lo que equivale a averiguar qué intervalo tiene un menor índice de solapamiento con el resto de los nodos de la partición. De esta manera, se trata de ir reduciendo la movilidad del punto de comienzo del nodo, hasta que este quede fijado en la posición de menor conflicto posible, para así maximizar el futuro reuso hardware.

Como se ha comentado anteriormente, dependiendo de la existencia o no de los intervalos I_a, I_b, así como de sus propias características, se estará ante distintos casos de particionamiento:

A)- No existe ninguno de los dos intervalos: Si esto ocurre porque el solapamiento de los rangos de *i* y *j* es nulo (Figura 4.8-a), el proceso de planificación del nodo *i* finaliza de una forma trivial, y se escoge su SS como punto de comienzo. Esto es así porque al elegir *j* como el nodo con mayor conflicto con *i*, se deduce que *i* no entra en conflicto con ningún nodo de la partición hardware, y por lo tanto el proceso de planificación no tiene trascendencia. Sin embargo, también puede ocurrir que la no existencia de los intervalos se deba a que el rango de *i* se solape totalmente con *j* (Figura 4.8-b). En este caso, se divide *rg(i)* en dos intervalos idénticos y se les asigna a I_a, I_b respectivamente. En esta situación pasamos a encontrarnos en el caso B)-.

B)- Existen ambos intervalos: En este caso, es preciso seleccionar aquel que suponga un menor solapamiento con respecto al resto de los nodos de la partición hardware. Para ello, se calcula el factor de conflicto global de cada uno de los intervalos, mediante la generalización de la expresión [4.24]:

$$\overline{\chi}_r = \sum_{k \neq i} \chi_{r,k} \qquad \textbf{[4.26]}$$

donde *r* se corresponde con el nodo *i* considerando cada una de las siguientes alternativas: eligiendo el rango de *i* como I_a o bien como I_b (Figura 4.8-c). El intervalo que ofrezca un menor índice de conflicto global es seleccionado, y se continúa en el punto C)-.

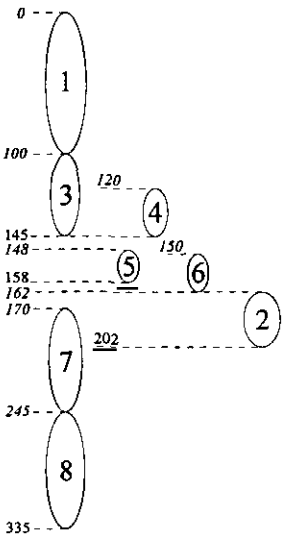
C)- Sólo existe un único intervalo: La consideración que hay que realizar entonces es si dicho intervalo *I* (Figura 4.8-d) es lo suficientemente grande para contener la ejecución del nodo *i*. En caso afirmativo, este intervalo pasa a ser el nuevo rango de *i*, *rg(i)*, y se vuelve al punto 2.- del algoritmo. Si no es así, se llega al caso base del proceso, en el que el nodo *i* ya no tiene movilidad, y por lo tanto queda fijado. Sólo queda escoger la posición relativa de *i* dentro de este intervalo. Se consideran dos posibles alternativas: hacer coincidir el punto de comienzo de *i* con el extremo superior del intervalo, o hacer coincidir el punto de finalización de *i* con el extremo inferior del intervalo. La planificación final, *p*, será aquella de las dos alternativas con un menor índice de conflicto global, $\overline{\chi}_p$.

4.- Después de la planificación de cada nodo es posible que para algunos de los restantes, sus tiempos SS y LS se hayan igualado, al haber sido forzados por las decisiones anteriores. En ese caso, la planificación de estos últimos se realiza de forma trivial.

5.- Repetir todo el proceso desde el punto 2.-, hasta que no queden nodos sin planificación.

La esencia de este método reside en la reducción gradual de los rangos de los nodos, empujando a estos progresivamente hacia zonas de más bajo conflicto, o hablando en términos de solapamiento, hasta que acaben siendo fijados, y por lo tanto, planificados.

Al final del proceso, la función objetivo expresada en [4.22] debería estar minimizada, lo que conllevaría el máximo aprovechamiento de los recursos hardware en el sistema.



Nodo	Tiempo
1	0
2	162
3	100
4	120
5	148
6	150
7	170
8	245

Figura 4.9

Tiempo de comienzo de cada nodo

Los resultados de planificación obtenidos para el ejemplo tratado en el presente capítulo aparecen reflejados en la Figura 4.9.

En este punto, cuando todos los nodos disponen ya de un tiempo de ejecución concreto, se tienen todos los requerimientos necesarios para proceder a calcular el coste global del sistema.

4.6 Modelo de estimación del coste.

El último paso del proceso de estimación consiste en calcular el coste global del sistema, lo que equivale a hallar este parámetro asociado al subsistema hardware. Esto es debido a que, como ya quedó explicado, el coste asociado a la parte software se considera nulo.

El coste del sistema se encontrará condicionado por la configuración particular de la solución examinada, a partir de las decisiones tomadas por el particionador. No sólo eso, el proceso de planificación macroscópica

realizado inmediatamente antes del cálculo de este valor, también ha podido influir sobre el parámetro, debido a su objetivo de minimizar el solapamiento.

En este punto, y según el sentido de la aproximación macroscópica, es preciso hallar cuánto hardware es posible compartir entre los distintos nodos existentes en la partición. Aunque ya se comentó en el punto anterior, recuérdese que esta capacidad de reuso debe ser directamente proporcional a la semejanza del par de nodos, e inversamente proporcional a su solapamiento, siempre en términos generales.

Así, se define el factor de reuso del nodo *j* respecto a las unidades funcionales del nodo *i*, y se representa $\rho_{j,i}$, de esta manera:

$$\rho_{j,i} = (1 - \sigma_{i,j}) \cdot \kappa_{[j,S(j)],[i,S(i)]} \qquad \textbf{[4.27]}$$

Es claro que esta definición cumple la relación entre sus parámetros según se ha definido previamente. Sin embargo, es obvio que no es la única expresión que *a priori* se puede proponer.

Otras funciones, en las que aparezcan distintos tipos de constantes, de factores exponenciados, u otra clase de operadores, como el cociente, también serían válidas. En este punto es preciso recordar la no consideración de unicidad sobre el sistema propuesto. Sencillamente, se trata de una representación del nivel macroscópico, adoptada como relevante, y que ha demostrado ofrecer unos buenos resultados.

El hecho de utilizar precisamente la expresión previa, es debido al principio de simplicidad, que ya fue mencionado anteriormente. Esta puntualización es extrapolable a otras expresiones que también serán presentadas. Pero este hecho no significa en absoluto que esta elección sea arbitraria, sino todo lo contrario, al haber sido elegida de entre otras posibles alternativas.

Se podría realizar un segundo inciso sobre el hecho de si sencillamente esta expresión, sin la consideración de la estructura interna de los nodos, es suficiente para representar el factor de reuso del hardware.

En otras palabras, puede surgir la cuestión acerca de si no se están obviando demasiados detalles en pro de la simplicidad, con una pérdida demasiado alta de exactitud en el sistema.

La respuesta sería la de siempre: obviamente, considerar todos los detalles ofrecería un mejor resultado, pero en un tiempo de cálculo mucho mayor y siempre que fuera factible[11]. Aunque pueda parecer lo contrario, en el nivel de trabajo en el que nos estamos moviendo, y con el tamaño de sistemas que se utiliza, los detalles de la estructura interna se desvanecen, dando paso a puntos de vista más generales.

De esta manera, podemos suponer un caso en el que dos nodos se solapen mínimamente, pero que sea justo en esa coincidencia mínima donde entren en conflicto sus unidades funcionales más costosas. Es evidente que ahí se produciría un error, ya que predeciríamos un reuso mucho mayor que el producido, debido a las particulares circunstancias de la distribución interna. Sin embargo, al tratar sistemas con un elevado número de operaciones, es seguro que esta circunstancia se presente en otros casos, pero con un error en diferente dirección: habrá veces en que se predecirá un reuso bajo entre dos nodos con un alto solapamiento, pero en el que no intervengan unidades similares, conduciendo a una compartición mayor de la esperada.

Si el problema es suficientemente grande, estos errores tienden a compensarse por motivos estadísticos. Esto es así porque no existe ningún factor que polarice los errores de este tipo en un sentido claro, o en otras

[11] Porque a veces la consideración explícita de todos los detalles no sólo ralentiza el proceso, sino que es probable que lo convierta en imposible.

palabras, no existe ningún motivo para que sobrestimemos el reuso la mayoría de las veces o viceversa. Por lo tanto, aunque se puedan obtener pequeñas variaciones alrededor del valor real, al ser estas de una forma oscilante, el resultado final tiende a aproximarse al verdadero.

Es normal que el comentario previo pueda parecer poco riguroso, ya que el dejar resultados importantes a merced del azar suele conllevar una pérdida de exactitud. Sin embargo, el hecho es que la teoría estadística funciona, y lo hace en este caso. No voy a negar que se está cometiendo un cierto error, ya que ninguna técnica de estimación está libre de imprecisiones, pero el error en este caso es de un orden de magnitud mucho menor del que *a priori* se pudiera sospechar, como se verá en el Capítulo 7.

Sin embargo, está claro que aunque este error no sea considerable, se podría disminuir aumentando la calidad de la técnica mediante una cierta consideración de la estructura interna. Si bien esto es cierto, hay que ser extremadamente cuidadoso con este planteamiento, ya que atravesar el límite del nivel de trabajo en el que nos movemos, considerando estructuras inferiores a los nodos, puede eliminar las virtudes de la técnica macroscópica presentada.

Además, es posible incrementar el tiempo de estimación en una proporción tal, que la técnica empiece a ser inviable para problemas de una cierta entidad. O bien que ese incremento de tiempo no compense el aumento de calidad conseguido, lo cual no sería un resultado deseable.

Por lo tanto, toda expansión del conjunto macroscópico ha de ser estrictamente controlada y sujeta a los principios del nivel explicados hasta ahora. Así, en el siguiente capítulo se presentarán unas mejoras a esta técnica básica como modelo para futuras expansiones.

Cerrados estos incisos, es preciso volver a la definición de reuso, dada por la expresión [4.27]. De esta manera, el coste de un sistema formado por estos dos nodos vendría dado por:

$$C = c_i^{S(i)} + (1 - \rho_{j,i}) \cdot c_j^{S(j)} \qquad \textbf{[4.28]}$$

Esta expresión tendría el significado de que el coste asociado al sistema sería igual al coste del primer nodo, con la implementación escogida por el particionador, más el coste del segundo nodo, bajo las mismas circunstancias, menos el coste que el segundo nodo ha sido capaz de reusar del primero, en forma de unidades comunes a ambos.

Es obvio que el coste hallado mediante la técnica de reuso es menor que el obtenido sumando sencillamente los costes de los nodos individuales. Sin embargo, es preciso proponer una extensión de la definición previa para hacer frente al coste total de un sistema con un número indeterminado de nodos.

La idea subyacente en este caso es la misma que la expuesta a lo largo de esta sección: reusar el mayor hardware posible con el fin de mantener el coste bajo. Así, el problema se reduciría a calcular el coste necesario para implementar un nodo *i* tras haber sido movido a la partición hardware, teniendo en cuenta que en ella ya se halla un conjunto de nodos $H = \{\alpha, \beta, ..., \psi, \omega\}$.

La aproximación simplista consistiría únicamente en sumar el coste de *i* al coste actual del sistema. Sin embargo, es bien seguro que parte de las funcionalidades de *i* coincidirán con algunas de las asociadas a los nodos del conjunto H. Por lo tanto, considerarlas de nuevo sería contraproducente, ya que existe la posibilidad de que puedan ser compartidas por distintos nodos.

El primer punto consiste en calcular todos los factores de reuso del nodo *i* con respecto a los nodos contenidos en H, $\{\rho_{i,j}\}$, $\forall j \in H$., mediante la expresión [4.27] comentada anteriormente. Sin embargo, como ahora se está tratando un problema con más de dos nodos, es necesario realizar una

corrección sobre los factores para que cumplan la condición de relación borrosa.

Efectivamente, todo factor ha de estar comprendido en el intervalo cerrado [0,1], lo que impide que el factor de reuso total del nodo *i* con respecto al conjunto H sea la suma de los factores individuales, ya que produciría probablemente un resultado fuera del rango mencionado.

Esto es debido a que el hardware que se puede reusar es siempre limitado, ya que el número de operaciones que debe realizar también lo es. Por lo tanto, hay que considerar el hecho de que el reuso de cualquier módulo con respecto a un conjunto tiene como límite el coste total del propio módulo.

Así, el nodo *i* intentará reusar el máximo hardware posible respecto del primer nodo del conjunto H, en este caso α. Posteriormente, sobre la parte de *i* que no ha podido ser compartida, se repite la operación con el siguiente nodo, β. Este proceso se va repitiendo secuencialmente intentando compartir el mayor hardware posible, siempre sobre la parte de *i* que no se ha podido reusar, hasta llegar al último nodo, ω. En este punto, las funcionalidades que no se hayan encontrado en H no tienen la posibilidad de ser reusadas de ninguna manera, y por lo tanto es preciso añadirlas al sistema. Es este hecho el que produce el incremento de coste no deseado, que en el mejor de los casos será nulo (cuando todas las funcionalidades de *i* han podido ser reusadas) y en el peor de los casos será el coste total del nodo (cuando ninguna de las funcionalidades se ha podido compartir).

Obsérvese que es precisamente este último caso, el peor de ellos, el que es utilizado por las aproximaciones clásicas que no tienen en cuenta el reuso, lo que dispara el coste del sistema enormemente cada vez que se añade un nuevo nodo a la partición hardware. Lo peor de esta técnica no es sólo esa sobrestimación del coste, sino que se aleja totalmente de

características físicas del hardware que deben ser consideradas si se quiere que la aproximación tenga algún vínculo con la realidad.

Otra conclusión que se puede extraer de la técnica de estimación de coste presentada, es que a medida que el tamaño del conjunto H crece, los factores de reuso de *i* con respecto a los últimos nodos van tendiendo a cero. Está claro que según avanza el proceso, *i* habrá conseguido compartir más funcionalidades, con lo que al final, el número de ellas que todavía queden libres irá disminuyendo progresivamente, y también las probabilidades de ser compartidas con otros nodos.

Por lo tanto, para hacer frente al estudio de los factores de reuso considerando la parte del nodo *i* ya compartida, se define el factor real de reuso de *i* con respecto a $j \in H$, $\rho_{i,j}^{real}$, el cuál es equivalente al definido en primer lugar, pero con la consideración mencionada. De esta manera, el cálculo de todos estos factores vendría dado por:

$$\rho_{i,\alpha}^{real} = \rho_{i,\alpha}$$

$$\rho_{i,\beta}^{real} = (1 - \rho_{i,\alpha}^{real}) \cdot \rho_{i,\beta}$$

$$\rho_{i,\gamma}^{real} = (1 - \rho_{i,\alpha}^{real} - \rho_{i,\beta}^{real}) \cdot \rho_{i,\gamma}$$

...

$$\rho_{i,\omega}^{real} = (1 - \rho_{i,\alpha}^{real} - \rho_{i,\beta}^{real} - \ldots - \rho_{i,\psi}^{real}) \cdot \rho_{i,\omega}$$

Se ve claramente en estas expresiones que los términos encerrados entre paréntesis son valores proporcionales a la parte del nodo *i* que todavía no ha sido compartida. Así, al multiplicarlos por el valor inicial del factor reuso, se obtiene su valor real final.

Otra observación iría encaminada a mostrar el progresivo descenso del reuso a medida que avanza este proceso. Efectivamente, cuando *i* trata de compartir hardware con los últimos nodos, ya quedan pocas de sus funcionalidades libres, y por lo tanto es más difícil encontrar unidades que

las implementen. Este hecho se observa en que los paréntesis de las expresiones son cada vez más amplios, indicando los sucesivos reusos previos que se han producido.

De esta manera, el factor total de reuso del nodo *i* respecto al conjunto de nodos hardware H, vendrá dado por la suma de todos los factores individuales:

$$\rho_{i,H}^{tot} = \rho_{i,\alpha}^{real} + \rho_{i,\beta}^{real} + \rho_{i,\gamma}^{real} + \ldots + \rho_{i,\omega}^{real}$$

Con esto se asegura que todos y cada uno de los factores de reuso están comprendidos en el intervalo [0,1], manteniendo con ello la coherencia del sistema.

Toda la formulación anterior se puede expresar de una forma más compacta mediante la utilización de sumatorios:

$$\begin{cases} \rho_{i,\alpha}^{real} = \rho_{i,\alpha} \\ \rho_{i,j}^{real} = (1 - \sum_{r=\alpha}^{j-1} \rho_{i,r}^{real}) \cdot \rho_{i,j}, \quad \text{si } j \neq \alpha \end{cases} \qquad \textbf{[4.29]}$$

$$\rho_{i,H}^{tot} = \sum_{j=\alpha}^{\omega} \rho_{i,j}^{real}$$

Así, el coste real que será necesario añadir al sistema para implementar las funcionalidades del nodo *i*, será inversamente proporcional a todo el coste que se pudo reusar. En consecuencia, se obtiene que:

$$c\_real_{i,H}^{S(i)} = (1 - \rho_{i,H}^{tot}) \cdot c_i^{S(i)} \qquad \textbf{[4.30]}$$

Obsérvese que este coste real asociado al nodo *i* con su implementación *S(i)* también tiene la dependencia de H, ya que es precisamente este conjunto el que determina cuánto hardware es posible reusar.

Hasta aquí he presentado el modelo básico que conforma la técnica de reuso macroscópica. Es importante notar que en ningún caso se ha descendido a un nivel inferior al del nodo, y que por tanto se han usado estrictamente datos y técnicas macroscópicas asociadas al nivel de trabajo.

Sin embargo, es preciso añadir una nueva consideración para satisfacer uno de los requerimientos asociados al concepto de reuso. Este requerimiento viene asociado al hecho de que una unidad funcional hardware sólo puede ser compartida en un cierto instante de tiempo si en dicho instante no está ocupada por ningún módulo del sistema.

Este es un hecho obvio, ya que de intentar usar una unidad por dos módulos simultáneamente, se produciría un conflicto estructural, y por supuesto una situación totalmente ilógica. Por lo tanto, es preciso dotar a la técnica de estimación del coste de esta información para que sea capaz de predecir estos casos de conflicto, y que, de esta manera, sólo considere un reuso de cada unidad por parte de un único módulo.

Para expresar la idea anterior de una forma más matemática, supongamos un nodo *j* que comparte ciertas unidades funcionales con otro nodo *i*. Este hecho produce el efecto colateral de que todo nodo *k* que se ejecute en paralelo con *j* sea incapaz de reusar esas funcionalidades concretas de *i*.

En un sentido más informal, aparentemente se produciría un descenso de la capacidad de reuso, y por lo tanto de la semejanza, de todo nodo *k* con respecto a *i*. Evidentemente, este descenso de la semejanza es ficticio, ya que como se comentó al principio del capítulo, dicho factor es intrínseco a cada par de nodos, y por lo tanto inamovible.

Sin embargo, la consideración de ese descenso puede valer para incorporar este hecho al modelo de estimación de coste. En este caso, habría que definir un nuevo factor ficticio de semejanza, cuyo valor inicial

fuese la semejanza real, y que se fuese decrementando a medida que se produzca el fenómeno descrito anteriormente.

Para llevar a cabo esta corrección, definamos primero la ocupación de las unidades funcionales del nodo *i* por parte del nodo *j*, representada por ω_i^j:

$$\omega_i^j = \rho_{j,i}^{real} \cdot \frac{\kappa[i,S(i)][j,S(j)]}{\kappa[j,S(j)][i,S(i)]} \qquad \textbf{[4.31]}$$

Este factor indica, mediante un número real perteneciente al intervalo [0,1], cuánto se reusa el hardware de *i* por *j*, desde el punto de vista del nodo *i*. Por lo tanto, es necesario pesar el factor de reuso mediante el cociente de las semejanzas entre *i* y *j*, y de esta forma obtener un factor de ocupación relativo a *i*. Es importante notar que la restricción de que ω ha de estar comprendido entre 0 y 1 es fundamental, ya que al estar constantemente trabajando con factores relativos, una violación de esta restricción haría perder consistencia al modelo.

Así, se ve en la expresión [4.31] que $\rho_{j,i}^{real}$ pertenece al intervalo [0,1], pero no está tan claro qué ocurre con el cociente de las semejanzas. Si bien cada uno de estos factores también cumple la restricción, al dividirlos no se puede afirmar nada *a priori* acerca de su resultado.

Sin embargo, si se observa la definición de $\rho_{j,i}^{real}$ en la expresión [4.29], y se sustituye en [4.31], se obtiene una definición equivalente para ω:

$$\left.\begin{array}{l} \text{(A): } \omega_i^j = \rho_{j,i}^{real} \cdot \dfrac{\kappa[i,S(i)][j,S(j)]}{\kappa[j,S(j)][i,S(i)]} \\ \text{(B): } \rho_{j,i}^{real} = R \cdot \rho_{j,i} \;;\; R \in [0,1] \\ \text{(C): } \rho_{j,i} = (1-\sigma_{i,j}) \cdot \kappa_{[j,S(j)][i,S(i)]} \end{array}\right\} \; \omega_i^j = R \cdot (1-\sigma_{i,j}) \cdot \kappa_{[i,S(i)][j,S(j)]} \qquad \textbf{[4.32]}$$

K	2	4	5	6
2	-	0.80	0.35	0.30
4	0.95	-	0.80	0.30
5	0.45	0.55	-	0.60
6	0.65	0.20	0.45	-

Σ	2	4	5	6
2	-	0	0	0
4	0	-	0	0
5	0	0	-	0.80
6	0	0	0.67	-

ρ^{real}	2	4	5	6
2	-	0	0	0
4	0.95	-	0	0
5	0.45	0.30	-	0
6	0.50	0.07	0.04	-

$c\_real_2^{HW_2} = 15$

$c\_real_4^{HW_1} = (1 - 0.95) \cdot 65 = 4$

$c\_real_5^{HW_3} = (1 - 0.45 - 0.30) \cdot 130 = 33$

$c\_real_6^{HW_1} = (1 - 0.50 - 0.07 - 0.04) \cdot 95 = 38$

Coste total = 15 + 4 + 33 + 38 =90

Figura 4.10

Resultados del cálculo del coste

El factor *R* que aparece sustituye al complemento a 1 de los factores de reuso parciales de *i*, según [4.29], y por lo tanto pertenece al intervalo [0,1]. Esto mismo ocurre con los otros dos factores de la expresión, el solapamiento y la semejanza. Por lo tanto, el producto de estas tres magnitudes estará por fuerza contenido en el mismo rango, y de igual manera, también lo estará ω. En consecuencia, el factor de ocupación es un elemento válido para los cálculos sobre magnitudes relativas que estamos llevando a cabo.

Retomando la corrección sobre la semejanza, recuérdese que su disminución en *k* (para todo *k* paralelo a *j*) respecto a *i*, será directamente proporcional al factor de ocupación de *i* por parte de *j*, y al grado de paralelismo de *k* con *j*. Así, se obtiene que la nueva semejanza ficticia obtenida viene dada por:

$$\kappa^{fic}_{[k,S(k)],[i,S(i)]} = \kappa_{[k,S(k)],[i,S(i)]} \cdot (1 - \omega_i^j \cdot \sigma_{k,j}) \qquad \textbf{[4.33]}$$

Este nuevo factor es el que se deberá utilizar para hallar el coste hardware de incorporación de sucesivos nodos. Con dicha corrección, se evitan los conflictos estructurales comentados anteriormente, y el coste obtenido puede calificarse de realista, al considerar las características físicas de paralelismo y compartición. Los resultados de la estimación de coste para el ejemplo presentado en este capítulo se pueden observar en la Figura 4.10.

Este método de estimación tiene la ventaja de poseer una complejidad temporal baja, comparada con otras técnicas microscópicas. Debido a que para calcular el coste de un nodo es necesario examinar el resto de los existentes en la partición hardware, la complejidad de este algoritmo puede verse acotada por $O(1 + 2 + ... + (n-1) + n) = O(n \cdot (n+1)/2) = O(n^2)$, donde n es el número de nodos hardware. Ya que el número de nodos que poseen un cierto grado de paralelismo es normalmente mucho menor que n, la complejidad de modificar κ, para hallar el valor ficticio de semejanza no es relevante comparada con la anterior.

Por lo tanto, el objetivo marcado previamente y relacionado con la necesidad de mantener el tiempo de ejecución en un nivel bajo, ha sido cumplido, permitiendo realizar el proceso de Codiseño mucho más rápido que siguiendo técnicas clásicas.

4.6.1. Importancia del reuso en el proceso de estimación.

Aunque pueda parecer que el hecho de considerar el reuso hardware es una suposición obvia, muchos de los entornos de Codiseño aparecidos ignoran completamente esta posibilidad[12], aplicando un modelo de estimación que se aleja de la realidad.

[12] Y no solamente los que aparecieron en primer lugar, sino también sus últimas versiones, así como entornos creados recientemente.

Cabe preguntarse acerca de cuánta precisión se pierde realmente al obviar la compartición de recursos, frente a aquellas técnicas que lo tienen en consideración. El hecho es que esta deficiencia se ve acentuada a medida que la partición hardware se va haciendo más costosa.

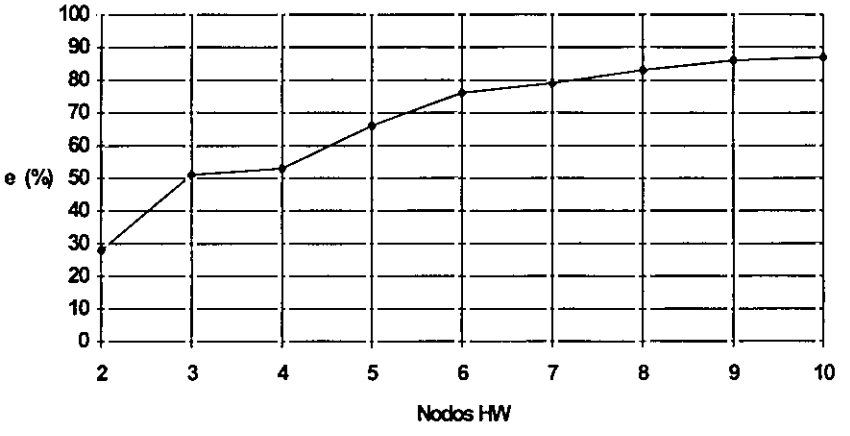
En efecto, si únicamente se dispone de un nodo hardware, parte de sus funcionalidades se verán reusadas al añadir uno nuevo, pero probablemente no sean todas, debido a las múltiples diferencias que pueden existir entre dos nodos. Sin embargo, cuando la partición hardware es suficientemente grande, el hecho de añadir un nuevo nodo no resulta en un exceso desmesurado de coste, ya que es probable que la mayoría de las nuevas funcionalidades puedan encontrarse entre la parte hardware ya existente.

En el caso extremo, se llegaría a la circunstancia de que el coste de añadir un nodo fuese nulo (en términos exclusivos de unidades funcionales). En otros palabras, que las nuevas funcionalidades añadidas fuesen un subconjunto de las ya existentes, pudiéndose implementar sin un esfuerzo adicional.

En consecuencia, a modo de resumen, podría decirse que la capacidad de reuso de un nodo aumenta gradualmente con la complejidad del sistema, la cual suele ser elevada en los casos de diseños reales, por lo que la no consideración de compartición hardware introduciría un error apreciablemente elevado.

El estudio que viene a continuación expresa cómo varía el incremento de coste de un sistema a medida que aumenta su número de nodos hardware, comparando los resultados con y sin la consideración de reuso.

Los experimentos se han realizado sobre 15 conjuntos de grafos generados aleatoriamente, con un número de nodos entre 5 y 20, y un número de nodos hardware entre 2 y 10. Cada conjunto contiene 5 grafos diferentes, lo que da un total de 80 resultados.



Nodos HW	c	c_r	e
2	80	58	27.5%
3	190	92	51.6%
4	305	143	53.1%
5	315	104	66.9%
6	415	99	76.1%
7	490	100	79.6%
8	560	90	83.9%
9	670	94	85.9%
10	740	95	87.2%

Figura 4.11

Efecto del reuso frente al coste

Cada grafo ha sido creado siguiendo la estructura y forma de problemas de Codiseño, de tal manera que los resultados sean extrapolables a casos reales.

Los datos obtenidos han sido el coste medio asociado a cada conjunto sin considerar reuso, c, considerando reuso, c_r, y la mejora obtenida al utilizar este último factor respecto al primero, lo que viene dado por el error relativo $e = 100 \cdot (c - c_r) / c_r$. Estos datos, así como una representación gráfica de los mismos, se pueden observar en la Figura 4.11.

Se ve claramente que a medida que el número de nodos hardware aumenta, la diferencia entre ambos modelos va siendo mayor, conclusión esta que coincide con la predicción hecha anteriormente.

En consecuencia, la utilización de técnicas de reuso está totalmente justificada, no sólo por los mejores resultados obtenidos, sino también por el bajo tiempo en que se consiguen.

Si no se emplease una aproximación macroscópica este tiempo sería considerablemente mayor, lo que podría obligar a no considerar el reuso para acelerar el proceso, y disminuiría por tanto la calidad de los resultados finales.

Expansión del Sistema Macroscópico: Las aproximaciones escalar y vectorial.

5

Tras haber explicado en el capítulo anterior la esencia del sistema macroscópico propuesto, y haber definido claramente las bases que configuran sus datos y sus métodos, el proceso de estimación de tiempo y coste para Codiseño queda fijado en el nivel de abstracción seleccionado.

Sin embargo, es conveniente dejar claro que el sistema presentado no es perfecto ni único, ya que una vez definido el entorno macroscópico, es posible llevar a cabo expansiones del conjunto de datos que afronten y solucionen detalles concretos de los problemas que se abordan. De esta manera, es factible añadir variables que estimen globalmente el número de registros, el interconexionado, o cualquier otra característica. Por regla general, si algo se puede calcular a nivel microscópico, existe un equivalente en el nivel macroscópico, por lo que las posibilidades de expansión son muy amplias.

No obstante, hay que enfatizar el hecho de que todas estas expansiones se han de llevar a cabo de una manera correcta y cuidadosa. En efecto, al añadir nuevos conjuntos de datos, es preciso respetar la idiosincrasia del nivel de trabajo macroscópico, lo que equivale a no bajar el grado de abstracción añadiendo información no relevante. De esta manera, lo que se quiere expresar es la inconveniencia de aumentar el nivel de

detalle para perfeccionar el sistema, por lo que la caracterización de nuevos aspectos del problema se debería realizar desde el mismo entorno propuesto para las metodologías del capítulo anterior.

Con el fin de ilustrar este proceso de expansión, voy a introducir en este capítulo un nuevo factor macroscópico para hacer frente a un problema concreto, que posteriormente se comentará. Al hacer esto, estaremos contemplando con un mayor grado de realismo el entorno de trabajo, lo cual no significa que la aproximación básica previa sea errónea, sino simplemente más generalista.

5.1 La estructura interna de los nodos.

Uno de los principales puntos de discusión sobre el modelo de estimación presentado es el siguiente: ¿Realmente tanta simplificación, al no considerar la estructura interna de los nodos, puede conllevar un grado de precisión adecuado de las soluciones?. Para afrontar esta cuestión, es necesario darse cuenta de que en el campo de la heurística, que es donde el mecanismo propuesto está encuadrado, no existe regla fija, sino que todo puede estar sujeto a consideración.

De esta manera, la respuesta a la pregunta anterior tendría una respuesta relativa: en efecto, el grado de precisión es adecuado (como se comprobará en los resultados experimentales), pero por supuesto podría ser mayor. Si los problemas tratados tienen la suficiente entidad, el modelo de estimación presentado no necesita considerar la estructura interna de los nodos. Los errores producidos por este hecho no estarán polarizados, es decir, la tendencia general del error no será a subestimar o sobrestimar el valor real de una forma continuada, sino a compensarse sucesivamente.

Esta no polarización, que hace que el valor estimado no se descompense de una manera progresiva, no evita sin embargo que el error

exista. Lo que ocurre es que ese error puede ser admisible, dependiendo de las circunstancias en las que nos movamos.

No obstante, se puede desear reducir esa deficiencia para aumentar el grado de precisión de las soluciones, por lo que entonces es necesario incrementar el grado de realismo de la aproximación, pero no bajando a un nivel de detalle inferior, sino caracterizando más detalles del problema.

Una de esas caracterizaciones debe ir dirigida hacia la consideración de la estructura interna de los nodos [MaMH99] [MaMo98a], que no quiere decir la consideración, tratamiento y proceso de las unidades funcionales presentes.

Por lo tanto, el introducir un nuevo parámetro macroscópico que sea capaz de representar este hecho de una manera eficiente contribuiría a aumentar el grado de precisión del sistema, sin que para ello sea necesario reducir el nivel de abstracción del entorno de trabajo.

5.1.1. El problema de la estructura heterogénea.

Tras lo comentado anteriormente, y enlazando esto con la definición de los métodos de estimación del capítulo anterior, se puede llegar a la conclusión de que contemplar los nodos de Codiseño como entidades carentes de estructura interna, puede considerarse un problema en sí mismo.

Recuérdese que la aproximación presentada proponía que la capacidad de reuso entre dos nodos era directamente proporcional a su semejanza, lo cual es obvio, e inversamente proporcional a su solapamiento, lo que ya no resulta tan claro.

Esta afirmación no es tan evidente porque bajo esa consideración, dos nodos que se ejecutasen en paralelo no podrían nunca compartir nada de hardware entre ellos. Sin embargo, puede darse el caso de que aunque exista dicha ejecución paralela, las unidades funcionales internas de ambos

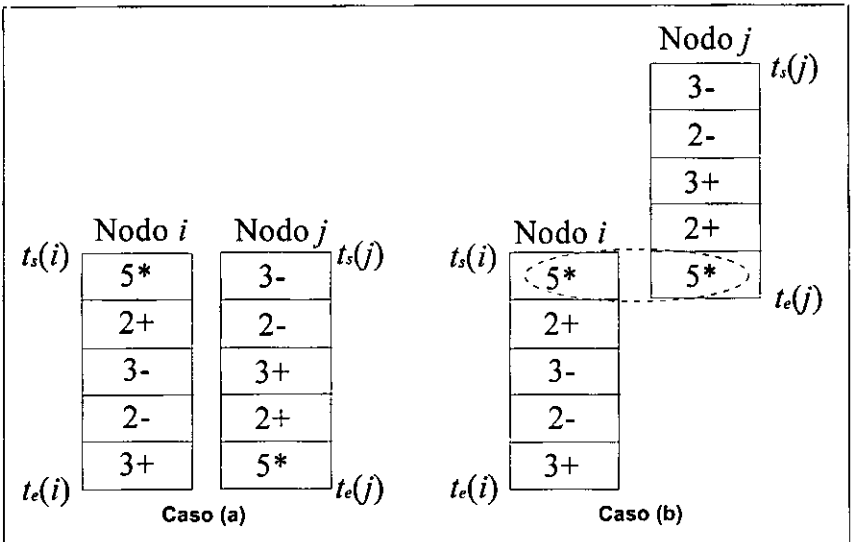


Figura 5.1

Distinta influencia del grado de paralelismo en el reuso

nodos no entren en conflicto, debido a que las operaciones de igual tipo tengan lugar en etapas de control no simultáneas.

De igual manera, si se tienen dos nodos con una ejecución casi secuencial, en la que únicamente coincida una etapa de control, sería previsible estimar que la capacidad de reuso tendería al máximo. No obstante, puede darse el caso de que en esa etapa de control coincidente estén situadas la mayoría de las unidades funcionales semejantes, y por lo tanto, que sea imposible compartirlas. Si además estas unidades son las más costosas, el reuso global de los dos nodos puede verse seriamente disminuido por este hecho.

Por lo tanto, existen casos en los que la consideración previa entre reuso y solapamiento no se ajusta a la realidad. Un caso como el comentado se puede ver en la Figura 5.1. Las dos situaciones corresponden exactamente a los mismos dos nodos (por lo que el factor de semejanza es

idéntico y no introduce ninguna diferencia en el reuso), y lo único que se varía es el grado de solapamiento entre ambos.

Considerando la aproximación básica, la ejecución paralela del caso (a) debería conllevar un reuso nulo, mientras que en el caso (b), en el que los dos nodos apenas coinciden, tendría un reuso cercano al máximo.

Sin embargo, es fácilmente comprensible que nos encontramos ante la situación descrita anteriormente, en la que las estructuras internas de los nodos tienen una importancia preeminente.

En efecto, debido a la disposición particular de los multiplicadores, que tienen lugar en etapas de control distintas en ambos nodos, el conflicto entre ellas es nulo en el caso de la ejecución paralela, y sin embargo existe una coincidencia total cuando se intentan ejecutar de una forma casi secuencial.

Por lo tanto, aquí se observa que la disposición interna de los nodos ha hecho que el reuso sea máximo en el primer caso, y que disminuya de forma considerable en el segundo.

Es cierto que este ejemplo con únicamente dos nodos no se ajusta bien a los problemas típicos de Codiseño, en los que la complejidad suele ser mayor. En esos casos, este error en la estimación del reuso se vería en parte compensada dentro de todo el proceso. Sin embargo, en este ejemplo concreto, la no consideración de la estructura interna ha llevado a una gran falta de precisión global.

5.1.2. La naturaleza intrínseca de los nodos.

El problema que surge en el ejemplo previamente comentado se debe a que los nodos, en su esencia, no son entes homogéneos, como se sugería en la aproximación anterior.

Al afirmar que no son objetos homogéneos, no me refiero a que no puedan ser modelados como tal[1], sino a que su esencia, como objeto real, es heterogénea.

Esta naturaleza intrínseca, si no es considerada, presentará más problemas en aquellos casos en los que los nodos ofrezcan muy distintas cargas operacionales. En otras palabras, si existen zonas de los nodos en los que manifiestamente se realicen un alto número de operaciones costosas, frente a otras áreas con una muy baja densidad de cálculo, los problemas que pueden aparecer serán mayores que en el caso de nodos más regulares.

Obviamente, no es necesario que el contenido de estos nodos deba estar sujeto a ningún tipo de distribución prefijada. De esta manera, no hay razón para pensar que la mayoría de los multiplicadores estarán agrupados, o que los sumadores aparezcan juntos en determinada posición.

Bajo esta consideración, y hablando desde un punto de vista global, podría entenderse una cierta homogeneidad en el sistema: los excesos de unidades en ciertas posiciones no tienen por qué ser extremados, y la distorsión en la regularidad que se comete se ve compensada por el resto del sistema.

Sin embargo, y aunque la aceptación de sistema homogéneo es perfectamente factible, se puede reducir el error introducido por esta mediante la consideración de la estructura irregular de los nodos.

[1] Lo cual ya se hizo en el capítulo anterior con buenos resultados.

5.2 La aproximación escalar.

Tras las apreciaciones previas de incorporar el factor de estructura interna, surge un nuevo acercamiento al proceso de estimación que denominaré *aproximación escalar*[2]. En los sucesivos apartados iré presentado cada una de las características que es preciso añadir a la técnica del capítulo anterior, a la que me referiré a partir de ahora como *aproximación plana*[3].

5.2.1. Modelado de la estructura: El parámetro densidad.

Una vez decidido que es preciso modelar los nodos como estructuras heterogéneas, cabe preguntarse acerca del método más apropiado para ello. Obviamente, y como ya se comentó antes, todo dato manipulado ha de estar contenido en el nivel macroscópico, por lo que no es posible trabajar directamente con el mapa de las unidades funcionales asociado a cada módulo.

De esta manera, lo que se pretende es caracterizar toda esa información con un nuevo parámetro que cumpla los requerimientos impuestos. Para ello, es preciso analizar cuál es la esencia del hecho que genera el problema tratado.

Esta esencia no es otra sino la diferencia de costes que conlleva implementar las distintas partes de un nodo. En otras palabras, el asumir que las distintas zonas de un nodo cuestan lo mismo introduce un error de apreciación que se puede manifestar después en el proceso de estimación de coste.

---

[2] Este nombre cobrará pleno significado a lo largo del capítulo.

[3] Este calificativo proviene del hecho de no considerar ningún tipo de estructura interna a los nodos.

Este hecho es el asociado básicamente a la idea de homogeneidad, la cual induce siempre a la conveniencia de reusar únicamente altos porcentajes de los nodos, sin tener en cuenta el coste real presente en cada caso.

En el conjunto de variables macroscópicas definidas en el capítulo anterior se incluyó, como es bien lógico, el coste relacionado a cada nodo, para cada una de las implementaciones escogidas.

Sin embargo, no se referenció en absoluto cómo se repartía ese coste a lo largo del rango de ejecución del nodo. Al no hacer esto, implícitamente se consideraba un reparto equivaluado y proporcional. Ahora, cuando ha quedado decidido que esa división constante no modela adecuadamente el comportamiento real del nodo, es preciso expandir esa noción de coste.

El problema básico reside en que *a priori*, cuando se está calculando este parámetro asociado del nodo para utilizarlo como entrada al proceso de Codiseño, los detalles provenientes de la planificación y asignación de hardware están presentes, pero es imposible reconstruir dichos detalles una vez iniciada la metodología con la sola ayuda del coste hallado.

Hay que recordar que una vez se ha comenzado con las técnicas de Codiseño, toda información interna de los nodos es transparente a estos procesos, con el fin de garantizar un tiempo de finalización aceptable. Por lo tanto, es necesario caracterizar este aspecto de las tareas de una forma más precisa, para que junto con el parámetro de coste, sea posible intuir la estructura interna en una posterior consideración.

Como lo que se va buscando es la diferenciación de las distintas etapas de los nodos según su coste de implementación, lo más lógico parece recopilar de alguna manera la tendencia de evolución de este coste dentro de todo el rango de ejecución.

En consecuencia, sería preciso caracterizar los distintos pesos de coste asociados a cada una de las etapas internas de los nodos, para así conocer las zonas de máxima y mínima densidad operacional.

Y es precisamente el concepto de densidad el que parece más adecuado para definir este nuevo parámetro. Esta afirmación es la que da pie para aclarar, antes de seguir adelante, un detalle conceptual clave: lo que se pretende no es conocer exactamente el coste asociado a cada etapa de control, equivalente a manipular directamente las tablas de unidades funcionales, sino comprender, desde el exterior, qué zonas son las que pueden provocar un mayor conflicto de unidades.

Esta aclaración viene ligada al hecho de que no se ha abandonado en absoluto la necesidad de tener el tiempo de diseño fuertemente controlado, así como el nivel de detalle admitido.

Por lo tanto, el nuevo parámetro de densidad, debe poseer las dos características siguientes, que resumen la consideración previa:

- Es preciso que no vaya asociado a una excesiva cantidad de información. En caso contrario, los datos que se deberían almacenar harían aumentar la complejidad espacial del problema, lo cual va sin duda unido a la complejidad temporal. No en vano, el tiempo asociado a la manipulación de un alto número de datos es un porcentaje significativo de la complejidad total, que debe mantenerse tan baja como sea posible.
- Los métodos que se apliquen sobre estos parámetros han de ser lo más simples posible, por la misma razón de no consumir un tiempo excesivo. Por lo tanto, el parámetro propuesto debe ser tal que operaciones como el cálculo del solapamiento no sean excesivamente complicadas. Esto descarta automáticamente la consideración de largas tablas de unidades funcionales, debido a la complejidad de cálculo sobre ellas.

Recopilando todos los requerimientos mencionados en los párrafos anteriores, se llega a la conclusión de que el parámetro densidad ha de ser

fácilmente almacenable, con métodos definidos sencillos, y que sea capaz de representar el coste asociado a cada instante de ejecución del nodo.

La última condición sugiere el hecho de que la densidad deba ser tratada como una función. Esto no supone ningún problema conceptual, ya que toda función no es sino la representación de una magnitud en distintos instantes.

Por lo tanto, a partir de ahora, se puede considerar al parámetro densidad como una correspondencia entre el espacio finito de las etapas de control de un cierto nodo, y el espacio de los costes de implementación.

Sin embargo, está claro que una tabla que represente para cada etapa su coste asociado, cumple a la perfección la proposición anterior, pero que sin embargo viola en su totalidad las otras dos premisas de simplicidad en el almacenamiento y en el cálculo.

Por eso desde un primer momento se ha negado la conveniencia de utilizar este tipo de estructuras, reflejo de la información microscópica de planificación y asignación. En consecuencia, es necesario dotar a la función densidad con alguna característica que permita solventar este tipo de carencias. Todas estas consideraciones parecen apuntar a la misma característica: la función densidad debe ser continua.

En un primer momento, esta afirmación puede causar sorpresa. No en vano nos estamos moviendo en sistemas puramente digitales, en los que la discretización es un hecho no discutible. Por lo tanto, cabe plantearse acerca del sentido de introducir entidades continuas.

Estas dudas, que si bien son totalmente razonables en el espacio microscópico, tienden a desaparecer cuando se evalúan los hechos desde el nivel de trabajo macroscópico. De nuevo es necesario un esfuerzo de integración en este espacio superior, abandonando las concepciones previas de niveles con más detalle.

Está claro que los nodos contienen un número finito de etapas de control, cada una de ellas con un número también finito de unidades funcionales. Pero si se consideran estos nodos, que ya de por sí tienden a ser complicados, dentro de la globalidad del sistema, se observa que se pierde ese nivel de detalle, y que los espacios discretos pasan a poseer características continuas.

Quizás sea esta la propiedad fundamental del nivel macroscópico, la que realmente diferencia este punto de vista de los más clásicos: el acercamiento a la continuidad de la información.

Una vez sentadas las bases acerca de cómo debe estar constituida la función densidad, que a partir de ahora denominaré δ, hay que escoger de entre todas las posibles estructuras continuas aquella que resulte más favorable.

Obviamente, y si lo que se quiere expresar es la tendencia de variación del coste, el principio de simplicidad sugiere la elección de una estructura polinomial para esta representación. Es claramente observable que estas funciones reúnen todas las características mencionadas anteriormente: son fáciles de almacenar (basta con un conjunto muy reducido de coeficientes) y los cálculos sobre ellas son muy sencillos (tales como el producto o la integración).

Por lo tanto, parece una decisión acertada el utilizar polinomios para expresar la densidad operacional de los distintos nodos. Es obvio que con esto no se consigue una exactitud total, pero tampoco es lo que se pretende. La idea reside en marcar la tendencia para delimitar las distintas zonas, no conocer con precisión el coste asociado a cada etapa de control.

De esta manera, el parámetro densidad pasa a formar parte del conjunto de variables macroscópicas, enriqueciendo la semántica del parámetro de coste. En consecuencia, una vez comenzado el proceso de Codiseño, la densidad pasará a representar la estructura interna de las

tareas, obviando toda referencia concreta a la distribución particular de unidades funcionales.

Es claro que existirá una función para cada nodo y para cada una de las implementaciones escogidas de este, por lo que la expresión concreta de esta función vendrá dada por $\delta_{i,S(i)}$.

El siguiente punto que se debe tratar es el método de construcción de estas funciones, a partir de la distribución original del nodo. Es conocido por distintos teoremas de Análisis Matemático, que toda función continua es aproximable a un polinomio con el grado de precisión que se requiera. Por lo tanto, el método más apropiado consiste en seleccionar un conjunto de puntos de la distribución inicial, y calcular un polinomio que pase por ellos. Con esto, y si los puntos son significativos, se marcarán las tendencias de crecimiento y decrecimiento del coste.

De esta manera, es preciso determinar cuál debería ser el criterio de selección de dichos puntos para que la función obtenida contenga la suficiente información, sin que llegue a ser excesiva la complejidad.

Dos puntos que parecen obvios son el máximo y el mínimo absoluto de la distribución, ya que si se pretende delimitar las etapas singulares, estos dos extremos deberían estar presentes. Así, se formaría un polinomio de orden tres, con únicamente cuatro coeficientes, lo que facilitaría sin duda su almacenamiento y manipulación.

Por supuesto, el hecho de incluir solamente dos puntos dejaría fuera de la representación otra serie de extremos locales, que no serían modelados por la función de densidad. Aquí es donde la decisión y experiencia del diseñador adquieren importancia, ya que se debe decidir entre el grado de detalle que se quiere reflejar y la complejidad final de la representación.

En consecuencia, si existen zonas características con extremos lo suficientemente significativos como para tener relevancia en los cálculos

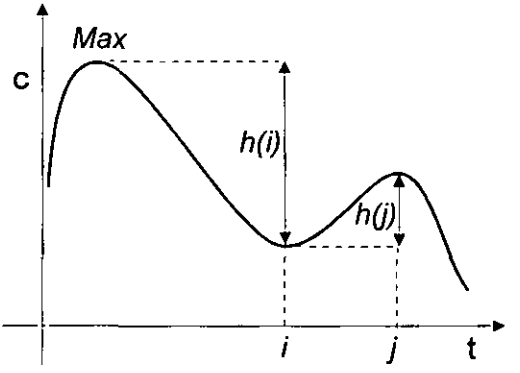


Figura 5.2

Altura relativa de los puntos

globales del coste, el conjunto de puntos que se ha de considerar debería aumentarse, formando un polinomio de orden mayor que tres.

Aquí se observa sin duda otra de las ventajas de esta aproximación: es muy versátil y adaptable a las necesidades concretas del momento. Así, es posible trabajar con polinomios muy sencillos, pero no obligatorio, ya que resulta fácil aumentar el nivel de precisión incrementando la complejidad de la función. Esta versatilidad contrasta fuertemente con el uso clásico de las tablas de unidades funcionales, en las que no cabía realizar ningún tipo de simplificación.

Como criterio general, se podría indicar que un extremo *i* (máximo o mínimo) debería ser considerado a la hora de construir el polinomio si su altura relativa, *h(i)*, es mayor que un cierto margen de tolerancia, τ, respecto al máximo absoluto.

La altura relativa en un punto se refiere al valor de la función en ese punto medido desde el extremo inmediatamente anterior (Figura 5.2). Así, un punto *i* se consideraría si cumple la relación:

$$\frac{h(i)}{MAX} > \tau \qquad \textbf{[5.1]}$$

donde *MAX* representa el valor del máximo absoluto. De esta manera, sólo se tendrían en cuenta los valores que representaran una variación significativa respecto a la tendencia general, obviando aquellas distorsiones no relevantes.

Un valor apropiado para el margen de tolerancia τ no debería ser inferior al 80%, con el fin de no considerar excesivos puntos que ralentizasen el proceso, aunque dicha decisión, en último término, radica en el diseñador.

Una vez decididos los puntos que se van a admitir, es posible determinar la expresión del parámetro densidad. Existen distintas herramientas comerciales destinadas al cálculo matemático, que permiten el ajuste de un conjunto de puntos por un polinomio. Este ajuste se realiza mediante complejos cálculos de aproximación, lo cual no tiene efectos sobre la metodología de Codiseño, ya que se realizan *antes* de que este proceso comience. Por lo tanto, todo el tiempo que se pueda emplear en un buen cálculo de los parámetros macroscópicos no repercute directamente en el tiempo de búsqueda de la solución.

Sin embargo, aquí se propone otro método, que si bien no es tan preciso, elude las dependencias de herramientas externas, y puede ser utilizado con bastante rapidez y facilidad.

Dicho método consiste en la resolución de un sistema de ecuaciones, tantas como puntos haya, y que permite calcular los coeficientes del polinomio que pasa por el mencionado conjunto de puntos. Además de esto, es conveniente incluir otras dos ecuaciones que delimiten la condición de extremo del máximo y del mínimo absolutos, para que la tendencia general del sistema quede claramente reflejada. Así, el sistema de ecuaciones descrito vendría dado por:

$$\begin{cases} \delta(x_i) = y_i \\ \delta'(x_j) = 0 \end{cases} \qquad \textbf{[5.2]}$$

donde (x_i, y_i) son todos los puntos del conjunto seleccionado (la primera ecuación es la condición de punto) y x_j representa el máximo y el mínimo absolutos (la segunda ecuación es la condición de extremo).

De esta manera, si n es el tamaño del conjunto de puntos seleccionado, se obtendrían n+2 ecuaciones, y el mismo número de coeficientes. Así, el polinomio final hallado sería de grado n+1.

Para el caso más sencillo de contar únicamente con los extremos absolutos, obtendríamos cuatro ecuaciones con cuatro incógnitas. Esto ofrecería un polinomio de orden tres, que evidentemente constaría de dos extremos (que son los que se han seleccionado previamente).

5.2.2. Nueva definición del factor de solapamiento.

Una vez definido el parámetro densidad, δ, es preciso revisar el objetivo final de esta expansión del conjunto macroscópico. Por supuesto, al verse alterada la consideración acerca de la estructura regular de los nodos del problema, el factor de solapamiento entre ellos ha de sufrir alguna alteración.

En consecuencia, es preciso redefinir este factor de solapamiento para que sea capaz de absorber las nuevas peculiaridades introducidas en el problema. De igual manera, si este factor se corrige, es de suponer que el coste final del sistema también sufrirá algún tipo de variación.

Precisamente es esta variación la que se buscaba en un principio, de tal manera que las posibles incidencias debidas a áreas de alta carga operacional se manifiesten, de una forma más real, en los resultados del proceso de estimación.

Por lo tanto, en esta aproximación deja de tener total validez la definición de solapamiento ofrecida en [4.10], debido a que esta considera únicamente la intersección física de los nodos, sin tener en cuenta su composición.

Así, será preciso introducir un nuevo concepto que sea capaz de ponderar el peso en coste de esta intersección, y no solamente su tamaño, como se comentó al comienzo de este capítulo.

Este concepto se puede resumir en la idea de que dos nodos se solaparán más, no cuanto mayor sea su parte coincidente, sino cuando lo hagan con más *intensidad*. Es precisamente este término el que tiene que concentrar el peso de las distintas estructuras internas, mediante la apropiada utilización de la función de densidad.

Si dos nodos tienen una coincidencia temporal en la que intervienen dos zonas con una alta carga computacional, su intensidad será elevada, y por lo tanto también su solapamiento. Por el contrario, si las zonas que intervienen poseen una baja densidad de operaciones, ambos parámetros asociados tenderán a un valor escaso.

De esta manera, un primer objetivo antes de llegar a la redefinición del solapamiento sería cómo expresar este concepto de intensidad, que se denominará ι, de una forma efectiva y consecuente con el resto de la teoría. Así, el encontrar una relación algebraica apropiada entre las funciones de densidad de ambos nodos, sería lo deseable con el fin de alcanzar esta meta.

Una primera propuesta para la intensidad entre dos nodos, al estar trabajando con el concepto de regiones coincidentes, sería asociar dicho parámetro al área de la intersección de ambas densidades.

De esta manera, si representamos ambos nodos por *i* y *j*, con una cierta implementación *S(i)* y *S(j)* respectivamente, su intensidad en la zona coincidente delimitada por el intervalo $[c_1,c_2]$ vendría dada por la expresión:

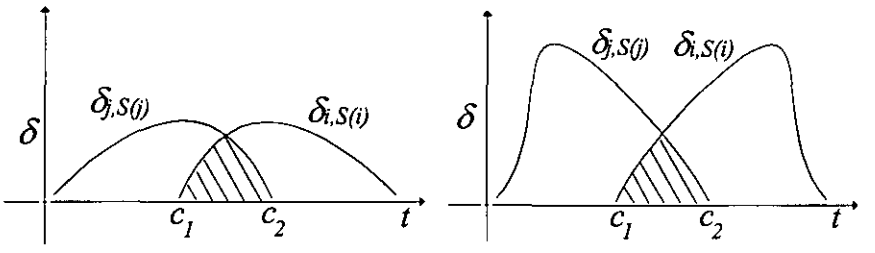


Figura 5.3

Caracterización de la *intensidad* como área de la intersección

$$\iota = \int_{c_1}^{c_2} min\left[\delta_{i,S(i)}(t),\delta_{j,S(j)}(t)\right]\cdot dt \qquad \textbf{[5.3]}$$

Esta expresión se ve ilustrada en el ejemplo de la Figura 5.3, donde la parte sombreada que aparece sería la representación de la intensidad. Obsérvese que ambas funciones están definidas en el dominio del tiempo, y por lo tanto también ocurre lo mismo con su integral.

Sin embargo, y aunque a primera vista pueda parecer lo contrario, esta definición de intensidad no es consecuente con lo que en un primer momento se intentaba representar.

En efecto, si comparamos ambos casos ofrecidos en el ejemplo, se comprueba fácilmente la no adecuación de la definición de intensidad.

Se observa que en los dos se ha obtenido exactamente el mismo valor para este parámetro, y sin embargo se trata de problemas con una característica diferenciada, intuyendo que el resultado que se debería hallar tendría que ser bien distinto.

Así, se ve en el segundo caso, que ambas funciones de densidad son más altas en la zona de intersección $[c_1,c_2]$. De esta manera, lo que se representa es que ambos nodos tienen más unidades funcionales operando

en dicha área, con lo que la posibilidad de conflicto, y por lo tanto su intensidad, debería ser mayor que en el primer caso.

No obstante, esto no se cumple, y el área de la intersección es idéntica en ambos ejemplos, por lo que la definición de intensidad no es válida: es preciso ajustarse fielmente a la idea que se pretende representar, o de lo contrario el modelo perderá su consistencia.

Esta mala interpretación se ha producido al asumir un error de concepto en el planteamiento. La cuestión reside en que, efectivamente, estamos tratando de calcular la posible intersección de dos conjuntos de unidades funcionales, lo cual no equivale a calcular la intersección de sus funciones de densidad.[4]

Realmente, este parámetro podría catalogarse como una probabilidad, que expresaría la facilidad de encontrar unidades funcionales en un cierto punto respecto al rango de ejecución global del nodo.

Por lo tanto, es más apropiado utilizar técnicas de tratamiento de probabilidades para conseguir expresar el concepto marcado. Repasando este último, se podría decir que lo que interesa es calcular la probabilidad de que unidades funcionales de ambos nodos entren en conflicto.

Si se observa la afirmación anterior, se puede intuir que lo más aconsejable sería calcular la función producto de ambas densidades (que marcaría la probabilidad de conjunción de ambos sucesos, en este caso de presencia de unidades de ambos nodos), y calcular su integral para extenderla a todo el área de intersección.

De esta manera, la nueva definición de intensidad entre dos nodos *i* y *j*, tal que *i* comienza su ejecución α unidades de tiempo después que *j*, vendría dada por:

[4] Lo que se podría intuir por la influencia de otros campos, como la lógica de conjuntos borrosos.

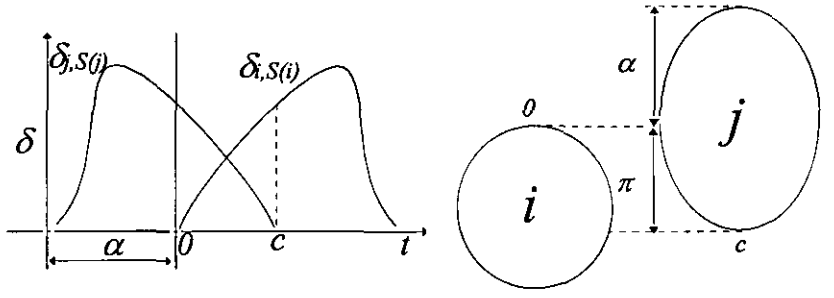


Figura 5.4

Caracterización de la *intensidad* como área del producto

$$\iota_{i,j}^{\alpha} = \int_0^c \delta_{i,S(i)}(t) \cdot \delta_{j,S(j)}(t+\alpha) \cdot dt \qquad \textbf{[5.4]}$$

La representación gráfica de esta expresión se puede observar en la Figura 5.4. Como se ve, la función de densidad de *j* está desplazada α unidades respecto de la primera. Esto es así para conseguir que ambos polinomios estén referenciados desde el mismo origen de coordenadas, en este caso, el punto de comienzo de la ejecución de *i*.

Por otra parte, se observa en la figura un nuevo factor, π, que tiene una significación física, correspondiente al grado de paralelismo existente entre ambos nodos, que, debido a la elección anterior de origen de coordenadas, vendría representado por el intervalo [0,c].

En este punto, cuando se ha decidido la caracterización final de la intensidad, se puede comprobar una de las mayores ventajas de introducir funciones continuas en el método de estimación: la facilidad de cálculo buscada desde un principio.

Efectivamente, la técnica para hallar la intensidad se ha reducido a la realización de las operaciones de producto e integración, las cuales resultan instantáneas al trabajar con polinomios. De esta manera, el tiempo final de

estimación no se verá excesivamente afectado por la introducción de este nuevo factor.

Una vez conseguida la representación de la intensidad, es posible establecer la nueva definición del solapamiento entre dos nodos, σ. El único problema que queda por resolver es la consideración de que σ debe ser un factor relativo, como ya se comentó en §4.1.5. No obstante, al no ser δ una función acotada, tampoco lo será ι, lo cual no es conveniente para este propósito.

Por lo tanto, y como el factor de intensidad es directamente proporcional al solapamiento existente entre los nodos, si se logra hallar una cota máxima válida para el primero, T, el segundo se podría redefinir así:

$$\sigma_{i,j} = \frac{\iota_{i,j}}{T} \qquad \textbf{[5.5]}$$

A la hora de calcular T, es preciso tener en cuenta que debe ser lo más ajustada posible al rango de valores de la intensidad. Por lo tanto, no sólo se busca una cota máxima, que puede no ser única, sino la mínima posible entre ellas.

Una posible expresión para T podría ser la siguiente:

$$T = K \cdot \int_{t_s(i)}^{t_e(i)} \delta_{i,S(i)}(t) \cdot dt \qquad \textbf{[5.6]}$$

donde los límites de la integral corresponden al tiempo de comienzo y finalización de la ejecución del nodo i, o en otras palabras, a los extremos del dominio de la densidad de i.

De esta manera, se tiene que la integral de $\delta_{i,S(i)}$ en su dominio, es siempre mayor o igual que su equivalente en la expresión [5.4], ya que esta última actúa en un intervalo más limitado (o a lo sumo igual)[5]:

[5] Esto es debido a que $[t_s(i), t_e(i)] \geq [0, c]$ siempre.

$$\int_{t_s(i)}^{t_e(i)} \delta_{i,S(i)}(t)\cdot dt \geq \int_0^C \delta_{i,S(i)}(t)\cdot dt \qquad \textbf{[5.7]}$$

La constante *K* que aparece en [5.6] representa la magnitud de la función $\delta_{j,S(j)}$. La pregunta que puede surgir aquí es por qué se ha elegido un *valor constante, y no una nueva integral, para que el modelo sea más* consecuente.

La razón para esto es bien clara, y está basada en el hecho de que se requiere que *T* sea del mismo orden de magnitud que la intensidad, como se propuso anteriormente. Si se añade una nueva integral para $\delta_{j,S(j)}$ sobre el rango de ejecución del nodo *j*, *T* estaría compuesto por el producto de dos integrales, mientras que ι vendría dada por la integral de un producto.

Aunque esta afirmación pueda parecer intranscendente, tiene un efecto muy relevante en los órdenes de magnitud de ambos parámetros. De esta manera, si consideramos que *n* es el orden del polinomio asociado a $\delta_{i,S(i)}$, y *m* es el equivalente para $\delta_{j,S(j)}$, y teniendo en cuenta que la integración de un polinomio da como resultado otro polinomio de un orden superior, se tiene que:

$$\begin{aligned} &\text{Grado}\{\iota_{i,j}\} = \text{Grado}\{\int_0^C \delta_{i,S(i)}(t)\cdot\delta_{j,S(j)}(t+\alpha)\cdot dt\} = 1+n+m \\ &\text{Grado}\{T\} = \text{Grado}\{\int_{t_s(j)}^{t_e(j)} \delta_{j,S(j)}(t)\cdot dt \cdot \int_{t_s(i)}^{t_e(i)} \delta_{i,S(i)}(t)\cdot dt\} = (1+n)+(1+m) \end{aligned} \qquad \textbf{[5.8]}$$

De esta manera, la definición de *T* estaría incorrecta, ya que sería un orden de magnitud superior a ι, lo cual no es deseable. Así, es más conveniente tomar el parámetro *K* como el valor absoluto máximo que $\delta_{j,S(j)}$ puede tomar en su dominio, por lo que se podría afirmar que:

$$\underset{dom(j)}{MAX}\,[\delta_{j,S(j)}(t)] \geq \delta_{j,S(j)}(t)\ ,\ \forall t \in dom(j) \qquad \textbf{[5.9]}$$

Por lo tanto, la expresión final de *T* vendría dada por:

$$T = \int_{t_s(i)}^{t_e(i)} MAX_{dom(j)} [\delta_{j,S(j)}(t)] \cdot \delta_{i,S(i)}(t) \cdot dt \qquad [5.10]$$

y considerando las desigualdades dadas en [5.7] y [5.9], se tiene que $T \geq t_{ij}$ siempre, concluyendo que la nueva definición de σ dada en [5.5] es válida.

5.2.3. Repercusión en el coste: Ejemplo de aplicación.

Evidentemente, y tras calcular una nueva expresión para el solapamiento, es de suponer que el proceso de estimación de costes ofrezca ahora ciertas variaciones en sus resultados.

Sin embargo, gracias a que el conjunto de parámetros macroscópicos se ha expandido de una forma consecuente, los nuevos conceptos son totalmente compatibles con las técnicas ofrecidas en el capítulo anterior.

Al hablar de una correcta expansión, no sólo me refiero a que los detalles internos a los nodos siguen sin considerarse una vez comenzado el proceso de Codiseño, sino a que el nuevo factor de solapamiento cumple los mismos requerimientos, en forma y significado, que el originalmente presentado: es un factor relativo, que relaciona pares de nodos, y con una concepción inversamente proporcional a la capacidad de reuso hardware.

De esta manera, el sistema macroscópico no ha sufrido ningún tipo de modificación en esencia, sencillamente ha cambiado uno de sus elementos de información. Por lo tanto, ningún método de los ya explicados debe sufrir cambio, ni ningún posible usuario del entorno tiene que realizar una reestructuración de contexto, ya que la única diferencia observable de la alteración producida es la variación de los resultados al estimar el coste.

Valga este capítulo como muestra a la hora de realizar futuras expansiones, tratando de mantener siempre la esencia contenida en el nivel de trabajo macroscópico.

En esta sección, trataré de mostrar con un ejemplo la ya mencionada variación en los resultados del coste al utilizar la nueva definición de solapamiento (aproximación escalar) frente a la presentada en el capítulo anterior (aproximación plana).

Para que el ejemplo sea ilustrativo, se estudiará un sistema reducido formado por dos nodos, *i* y *j*, con todos los posibles grados de paralelismo entre ellos. De esta manera, se comenzará con una ejecución secuencial de ambos, y se irá variando su posición relativa (indicada por el parámetro π), pasando por una coincidencia máxima, hasta llegar de nuevo al caso de ejecución secuencial (véase Figura 5.4).

Al final, se podrán observar las gráficas obtenidas para el solapamiento y el coste, así como el estudio de las conclusiones realizado. Con el fin de clarificar aún más el ejemplo, se mostrarán todos los cálculos detallados para un caso concreto de paralelismo entre ambos nodos.

Por lo tanto, y comenzando con la técnica de estimación, los pasos que hay que seguir son los siguientes:

- Selección de los dos nodos de Codiseño, *i* y *j*. A fin de simplificar, se supone una única implementación para cada uno. Cálculo, para cada uno de ellos por separado, de los parámetros macroscópicos intrínsecos de tiempo de ejecución, coste, función de densidad y semejanza.
- Determinación de una posición relativa entre ambos, lo que dará el nivel de paralelismo escogido.
- Cálculo del solapamiento entre ambos nodos, mediante las dos técnicas sometidas a estudio: primero, mediante la aproximación plana, sin considerar la estructura de los nodos; segundo, con la aproximación escalar, utilizando el factor de densidad hallado.
- Cálculo del coste de implementación de ambos nodos en conjunto, considerando el posible reuso hardware entre ellos, mediante las dos técnicas estudiadas.

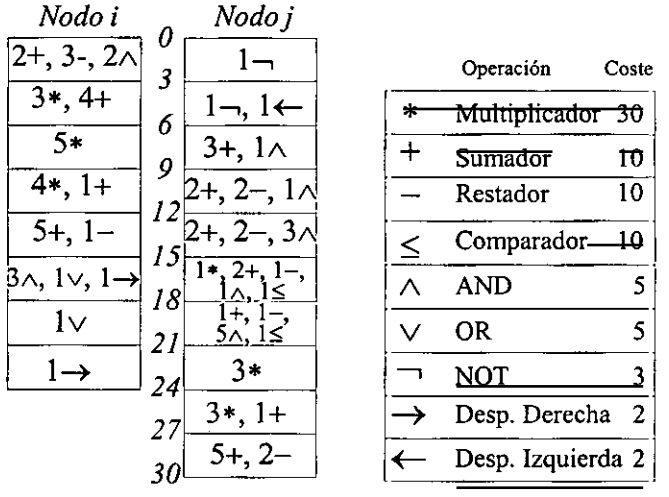


Figura 5.5

Ejemplo: Dos nodos coincidentes y tabla de unidades funcionales

- Comparación de los resultados obtenidos con el coste *real*, hallado mediante la aplicación de la tabla de unidades funcionales internas a los nodos (detalles microscópicos).
- Repetición del proceso, variando la posición relativa de ambos nodos, con el fin de barrer todo el espectro de posibilidades.

De esta manera, y como primer paso, se presentan los dos nodos que conforman el sistema, con la planificación de operaciones ofrecida en la Figura 5.5. Como se ve, existe una unión clara entre unidades funcionales y etapas de control.

El periodo de reloj se ha escogido como 3 unidades de tiempo. De esta manera, el nodo *i* tendrá un tiempo de ejecución de 24 unidades de tiempo, y el nodo *j* de 30.

El coste asociado a cada una de las unidades funcionales presentes se puede observar en la figura anterior. Así, y estudiando el grado de paralelismo de cada uno de estos tipos de unidades en cada etapa de

control, se puede calcular el coste de cada uno de los nodos por separado. Estos valores han resultado ser 252 y 200 unidades de coste para los nodos *i* y *j* respectivamente.

Para hallar la semejanza del nodo *j* respecto del nodo *i*, es preciso seguir su definición. Así, se calcula el hardware común a ambos nodos, lo que se puede realizar con un simple estudio de la tabla de unidades funcionales, ofreciendo un valor de 175 unidades de coste. Por lo tanto, para hallar la semejanza, basta dividir este valor por el coste total del nodo *j*, obteniendo que $\kappa_{[j,S(j)],[i,S(i)]} = 0.8750$.

Tras este punto, hay que determinar la posición relativa de ambos nodos. Para realizar los cálculos de una manera detallada, se ha elegido el grado de paralelismo como π=15 (Véase Figura 5.4). Esta elección hace que el factor de desplazamiento de ambos orígenes de coordenadas, α, también *sea igual a 15.*

Ahora, con todos los cálculos preliminares realizados, es posible proceder directamente a hallar el solapamiento mediante ambas aproximaciones:

a) *Aproximación plana.* Aquí usaremos la primera definición del solapamiento, en la que el único factor decisivo es el tamaño de la intersección de ambos nodos. La expresión así considerada se puede representar de la siguiente manera, con los datos del ejemplo, teniendo en cuenta que t_s y t_e son los tiempos de comienzo y finalización de los nodos respectivamente:

$$\sigma_{i,j} = \frac{t_e(j) - t_s(i)}{t_e(i) - t_s(i)} = \frac{15}{24} = 0.6250$$

b) *Aproximación escalar.* Ahora, es preciso considerar la estructura interna de ambos nodos, para estimar cuáles serán las áreas de máxima intensidad en el solapamiento. El primer paso consiste en calcular el coste asociado a cada etapa de control interna a los nodos. Esto se realiza de una forma

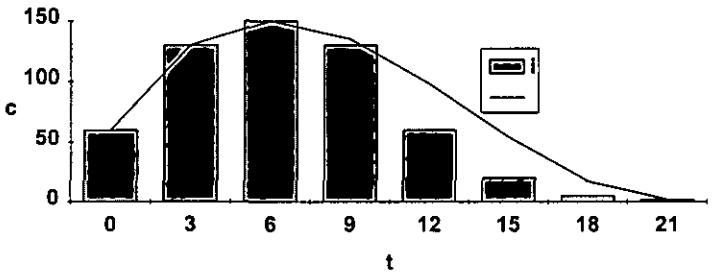


Figura 5.6

Estructura del nodo *i*

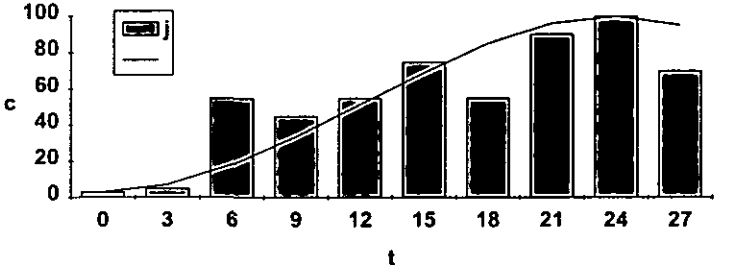


Figura 5.7

Estructura del nodo *j*

directa con una simple observación de las tablas de unidades ofrecidas en la Figura 5.5. El resultado de esta operación se puede observar en las gráficas de la Figura 5.6 (nodo *i*) y de la Figura 5.7 (nodo *j*), representado mediante la forma de columnas sombreadas.

Como se puede comprobar mediante el estudio de estos datos, se ha elegido la estructura del nodo *i* para que sea fácilmente aproximable mediante un polinomio de orden 3, mientras que se ha construido el nodo *j* con una estructura mucho más irregular. De cualquier forma, se ha optado por un conjunto de dos puntos (el máximo y el mínimo absolutos) para ajustar los polinomios asociados a ambos nodos.

Estos puntos son {(6,150),(21,2)} para el nodo *i*, y {(0,3),(24,100)} para el nodo *j*. Por lo tanto, el polinomio obtenido en ambos casos tendrá una estructura de grado 3. A la hora de elegir estos puntos se ha tenido en cuenta que ninguno de los restantes admitía un margen de tolerancia, τ, de al menos un 80%, por lo que se ha preferido obviarlos.

Las dos funciones obtenidas están representadas en las Figuras 5.6 y 5.7, mediante una línea continua en ambos casos. Las expresiones matemáticas para ambas, una vez obtenidos sus coeficientes mediante la resolución del sistema de ecuaciones [5.2], son las siguientes:

$$\delta_{i,S(i)} = 0.0877 \cdot x^3 - 3.5520 \cdot x^2 + 33.1520 \cdot x + 60.0160$$

$$\delta_{j,S(i)} = -0.0140 \cdot x^3 + 0.5052 \cdot x^2 + 3$$

Ahora, el valor del factor de intensidad se calcula mediante la expresión [5.4], que considerando los datos propuestos toma el valor de:

$$\iota_{i,j}^{15} = \int_0^{15} \delta_{i,S(i)}(t) \cdot \delta_{j,S(j)}(t+15) \cdot dt = 160259.6$$

Posteriormente, se halla el valor de la cota máxima *T* mediante [5.10]:

$$T = \int_0^{24} 100 \cdot \delta_{i,S(i)}(t) \cdot dt = 189506.5$$

Con estos resultados, el valor del solapamiento calculado siguiendo la expresión [5.5] es:

$$\sigma_{i,j} = \frac{\iota_{i,j}}{T} = 0.8456$$

El siguiente paso propuesto consiste en el cálculo del coste asociado a ambos nodos, mediante la utilización de las técnicas previas. En ambos casos, la expresión utilizada es la ofrecida en [4.28].

Se puede comprobar que ya se dispone de todos los datos necesarios para proceder al cálculo de dichos costes, obteniendo unos valores de:

$$C_{plana} = 252 + (1 - 0.3281) \cdot 200 = 387$$

$$C_{escalar} = 252 + (1 - 0.1351) \cdot 200 = 425$$

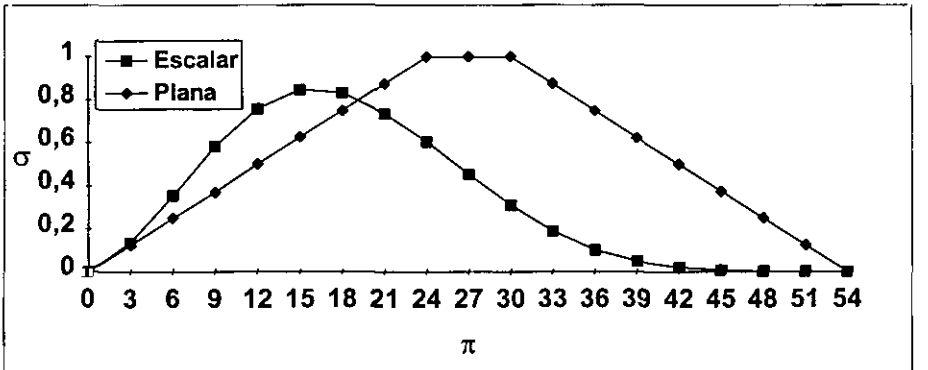


Figura 5.8

Resultados del solapamiento

En este punto es preciso hallar el coste real, considerando los detalles de planificación y asignación de hardware. Este proceso es directo, al disponer de toda la información necesaria acerca de las unidades funcionales (véase Figura 5.5). De esta manera, se obtiene un coste real de 427 unidades.

Se observa claramente que el coste ofrecido por la aproximación escalar se acerca más al real que el hallado mediante la aproximación plana. Sin embargo, no es posible generalizar esta conclusión, debido a que este fenómeno sólo ha sido probado para un caso de paralelismo concreto entre los dos nodos.

Por lo tanto, es interesante repetir la totalidad del experimento para todas y cada una de las posibles posiciones relativas de ambos nodos. Para conseguir esto, se hace que π varíe en el intervalo [0,54]. Las estimaciones asociadas a ambos extremos son casos con una ejecución secuencial, mientras que los valores intermedios indicarían distintos grados de paralelismo.

De esta manera, se puede observar en la Figura 5.8 los resultados del solapamiento por ambas técnicas. Como se puede ver, en el eje de abscisas

se representa el grado de paralelismo entre ambos nodos, mediante el factor π.

Hay algunas puntualizaciones que hacer a la vista de estos resultados:

- La gráfica de la aproximación plana es claramente simétrica. Este es un hecho razonable, ya que si se considera su definición, el solapamiento será máximo cuando se obtenga la más amplia intersección física de ambos nodos. Esto se producirá en el caso de mayor paralelismo, que es precisamente el punto medio de la gráfica, equidistante de las dos ejecuciones secuenciales.
- La aproximación escalar obtiene una gráfica cuyo máximo está desplazado hacia la izquierda, en contraste con el caso anterior. Aquí es donde se pone de manifiesto el efecto de considerar la estructura interna de los nodos. Es en este punto, el que corresponde al máximo, donde las unidades funcionales más costosas entran en conflicto, como se puede observar estudiando la tabla de distribución detallada, y así es como se ve reflejado en los resultados.
- El factor de solapamiento de la aproximación escalar nunca alcanza el valor de 1, al contrario de lo que ocurre en la aproximación plana. Este es un resultado razonable, ya que cabe preguntarse cuál es el sentido de obtener un solapamiento con el valor máximo. Dicho valor debería obtenerse únicamente en el peor caso, en el que todas las unidades funcionales de ambos nodos entrasen siempre en conflicto para cada una de las etapas de control, la cual es una situación extraña y nada habitual.

Una vez examinados los factores de solapamiento, comprobemos cómo han afectado estos al cálculo global del coste hallado por ambas técnicas, comparándolos con el coste real considerando la distribución de unidades funcionales. Las gráficas resultantes se pueden observar en la Figura 5.9. Los comentarios pertinentes acerca de estos resultados son:

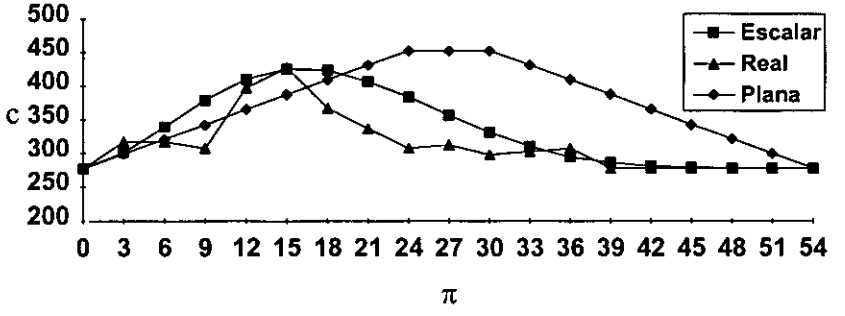


Figura 5.9

Resultados del coste

- La aproximación macroscópica encuentra resultados más reales que la plana en la gran mayoría de los casos. Además, estos resultados son más fiables, lo cual tiene gran importancia si se considera que el propósito de estas estimaciones es discernir entre la calidad de dos soluciones distintas cuando se realiza el proceso de particionamiento.
- El coste máximo real está más próximo al calculado por la técnica escalar que al hallado mediante la plana. Esto induce a pensar que el haber obtenido una gráfica del solapamiento desplazada es muy razonable, ya que se acerca al caso real, y por lo tanto el hecho de usar los polinomios propuestos es muy aconsejable.
- Cuando se trata de los casos de paralelismo nulo, o lo que es lo mismo, de ejecución secuencial, correspondiente a los puntos (π=0, π=54), ambos métodos encuentran el coste real. En estos casos, el único factor influyente que toma parte en el cálculo del coste es la semejanza (ya que el solapamiento es nulo). De este hecho se puede extraer la conclusión de que dicho factor está bien ajustado, al conducir al proceso a la consecución del mejor resultado.
- Aunque las gráficas asociadas al coste real y al hallado por la técnica escalar tienen la misma forma, y por lo tanto son fieles, existe una

diferencia de magnitud entre ellas en el intervalo (18,27). Esto puede deberse a la aproximación no perfecta de las tablas de unidades funcionales mediante los polinomios, aunque no puede considerarse como una carencia, ya que desde un primer momento se pretendió marcar las tendencias de los costes y no su valor exacto. Por lo tanto, el hecho de que ambas gráficas tengan la misma inclinación es una consecuencia lógica de la técnica escalar utilizada.

5.3 La aproximación vectorial.

En la sección anterior se ha presentado una expansión al conjunto inicial de variables macroscópicas, con el fin de considerar un nuevo detalle del sistema: la estructura interna de los nodos. Esto se consiguió mediante la introducción de un nuevo parámetro, la densidad, que caracterizaba de una manera global la tabla de planificación y asignación de unidades funcionales, pero de una forma más compacta.

Sin embargo, puede surgir la duda de por qué esta expansión se ha realizado de esta manera, y no se han considerado otras circunstancias u otros factores, igualmente intuitivos, y con posibilidad de ofrecer iguales o mejores resultados.

En lo que resta de capítulo comentaré otra posibilidad de expansión también válida, tomando como referencia la aproximación escalar, destacando las ventajas y desventajas existentes entre ambas.

5.3.1. Desdoblamiento de la función de densidad.

Una de las principales objeciones que se pueden lanzar a raíz de la elaboración de la aproximación escalar reside en que si optamos por el análisis previo de la tabla de unidades, con el fin de construir el parámetro densidad, por qué no hacerlo estudiando por separado cada tipo de operación.

De esta manera, puede parecer lógico generar una función independiente para cada uno de estos tipos de unidades funcionales, por lo que en vez de tener finalmente un único parámetro, obtendríamos una colección de ellos.

Esta técnica parece contener mayor dosis de precisión que la escalar, ya que se están tratando más detalles. Además, la separación clara entre los distintos tipos puede suponer una ventaja añadida, ya que así no existirá ningún género de duda en el momento de hallar los posibles conflictos de unidades.

Es obvio que de esta manera dichos conflictos se pueden calcular de un forma muy directa, lo cual no ocurría anteriormente. En efecto, cuando disponíamos únicamente de una sola función, al localizar una zona de alta carga operacional nos resultaba imposible discernir qué tipo de unidades la conformaban. Lo único que se podía afirmar es que el coste de esa zona era elevado, y por lo tanto que no era conveniente solaparla con otro nodo, pero la estructura concreta nos era desconocida.

Por lo tanto, y con el fin de compararla con las técnicas precedentes, se ofrecerá un estudio de una aproximación en la que se considere una función de coste aislada para cada uno de los tipos de operaciones presentes. Denominaré esta técnica como *aproximación vectorial*, al contar con un conjunto de parámetros, en contraste con la escalar, que sólo disponía de un único factor de densidad.

La primera consideración que hay que realizar acerca de la nueva técnica, es que por fuerza se incrementará el tiempo global de cálculo de las estimaciones. Obviamente, al manipular ahora más información, los procesos se hacen más complejos. Sin embargo, no parece que dicha complejidad vaya a aumentar drásticamente, sino que más bien debe afectar de una manera lineal.

Esto es debido a que el único cambio que debe sufrir la aproximación es la repetición del tratamiento de la densidad tantas veces como funciones haya, en lugar de la única vez que se efectuaba previamente. De nuevo aparece aquí el concepto de expansión controlada, en el que los métodos propuestos deben sufrir el menor número posible de cambios, no sólo por facilidad de integración, sino también por coherencia en las ideas básicas del sistema.

Una última consideración de gran importancia conceptual acerca de esta técnica: al tener ahora disponible la información concreta de cada unidad funcional, y por lo tanto, su distribución dentro de cada uno de los nodos del sistema, el parámetro macroscópico de semejanza se vuelve redundante.

En efecto, recordemos que dicho parámetro calibra cuánto hardware es compartible entre dos nodos, en proporción al coste total de ellos. Sin embargo, este dato se puede calcular ahora, conociendo en detalle la disposición de unidades contenidas en ambos nodos.

No obstante, no se pretende afirmar que la semejanza haya perdido su significación dentro del sistema, sino que al introducir nueva información, los métodos propuestos de estimación se pueden llevar a cabo de una manera más directa.

Por lo tanto, y a modo de resumen, el paso de la técnica escalar a la vectorial supone un cambio conceptual de comprensión de la información. Así, la semántica contenida en el parámetro de semejanza hace que la función densidad única se disgregue en un conjunto de funciones, que caracterizan cada nodo de una forma más explícita.

5.3.2. Adaptación del modelo matemático a la aproximación vectorial.

Es obvio que la introducción de nuevos parámetros hace que el modelo matemático que servía de base al proceso de estimación de costes sufra modificaciones para adaptarse a las nuevas circunstancias.

La primera novedad es, como ya se ha venido advirtiendo a lo largo de esta sección, que el parámetro densidad pasa de ser único a estar constituido como un conjunto de tantos elementos como tipos de unidades haya:

$$\bar{\delta}_{i,S(i)} = (\delta^1_{i,S(i)},\ldots,\delta^f_{i,S(i)}) \qquad \textbf{[5.11]}$$

donde *f* es el cardinal del conjunto *F*, formado por las unidades funcionales consideradas en el diseño. Obviamente, ahora existirá un vector[6] densidad para cada nodo, y para cada una de las implementaciones escogidas.

De igual manera, y si la densidad pasa a tener forma de vector, lo mismo ocurre con el solapamiento, ya que este último es resultado de operaciones realizadas sobre la primera. Por lo tanto, ya no es posible hablar del solapamiento entre dos nodos, en términos genéricos, sino que es preciso referirse al solapamiento entre un tipo particular de unidades funcionales, pertenecientes a dos nodos.

Aparte de esto, la expresión matemática que denota a este parámetro no sufre cambios. Así, dados dos nodos, *i* y *j*, para la *k*-ésima unidad funcional se obtendrá que:

---

[6] Aunque se denomine vector a esta magnitud, está claro que no lo es en el sentido algebraico de la palabra, ya que de ninguna manera pertenece a un espacio vectorial. La razón de este apelativo, así como su expresión, responden a criterios de familiarización e intuición.

$$\sigma_{i,j}^{k} = \frac{\iota_{i,j}^{k}}{T^{k}},\ \forall k \mid 1 \le k \le f \qquad \textbf{[5.12]}$$

Como se observa, esta definición es totalmente simétrica a la expresión [5.5], con la distinción de que aquí aparecen *f* componentes diferenciadas. De igual manera, se pueden expandir las expresiones [5.4] y [5.10], relativas a la intensidad ι y al umbral máximo *T*, quedando de la siguiente forma:

$$\iota_{i,j}^{k} = \int_{0}^{c} \delta_{i,S(i)}^{k}(t) \cdot \delta_{j,S(j)}^{k}(t+\alpha) \cdot dt,\ \forall k \mid 1 \le k \le f \qquad \textbf{[5.13]}$$

$$T^{k} = \int_{t_s(i)}^{t_e(i)} MAX_{dom(j)}[\delta_{j,S(j)}^{k}(t)] \cdot \delta_{i,S(i)}^{k}(t) \cdot dt,\ \forall k \mid 1 \le k \le f \qquad \textbf{[5.14]}$$

Pasemos ahora a estudiar cómo ha cambiado el mecanismo de estimación del coste. Antes de ello, es preciso darse cuenta de una característica muy importante que aparece en la aproximación vectorial.

Aunque por supuesto, las funciones de densidad sólo marcan la tendencia de aparición de las distintas unidades funcionales en los nodos, podemos conocer con exactitud el coste destinado a implementar cada una de ellas. Este hecho se produce debido a que este coste viene dado por la suma de unidades presentes en la etapa de máximo paralelismo.

En efecto, el número de unidades necesitadas por un nodo será el equivalente al máximo número de operaciones que se deben ejecutar en paralelo. A partir de esas unidades, y reusándolas apropiadamente *en el interior del nodo*, es posible implementar todas las operaciones presentes. Este concepto no debe aportar mucha dificultad, ya que es una de las bases de la Síntesis de Alto Nivel.

Por lo tanto, si se conoce el coste asociado a la etapa de máximo paralelismo de una unidad dada, se conocerá el coste total necesario para implementar todas las operaciones de ese tipo dentro del nodo en cuestión.

Pero ese coste es bien conocido, ya que al buscar la etapa de máximo paralelismo, nos estamos refiriendo al máximo coste absoluto presente en la tabla de distribución de esa unidad dada, siendo ese punto uno de los escogidos para construir la función densidad asociada.

Resumiendo, el máximo absoluto de cada una de las funciones de densidad δ, corresponde al coste total, *cf*, necesario para implementar las unidades funcionales correspondientes del nodo *i*:

$$cf^{k}_{i,S(i)} = \underset{dom(i)}{MAX}(\delta^{k}_{i,S(i)}),\ \forall k \mid 1 \le k \le f \qquad \textbf{[5.15]}$$

Así, conociendo este dato, el cálculo del coste del sistema se simplifica bastante. De esta manera, dados dos nodo *i* y *j*, y un tipo de unidad funcional común, *k*, el máximo coste que *j* puede reusar de *i*, respecto a esa unidad vendrá dado por:

$$\hat{cf}^{k}_{j,i} = MIN(cf^{k}_{j,S(j)}, cf^{k}_{i,S(i)}) \qquad \textbf{[5.16]}$$

Esta afirmación es bien clara, ya que no se puede compartir más de lo que necesite ninguno de los dos nodos. Esto sería, por supuesto, en el caso ideal de que las unidades de ambos nodos no entrasen en conflicto. Sin embargo, como siempre, si existe solapamiento las posibilidades de reuso disminuyen.

De esta manera, se define el coste real que dos nodos pueden compartir, asociado a una unidad funcional concreta, como un factor directamente proporcional al coste máximo posible, e inversamente proporcional al solapamiento entre los nodos:

$$cf^{k,real}_{j,i} = \hat{cf}^{k}_{j,i} \cdot (1 - \sigma^{k}_{i,j}) \qquad \textbf{[5.17]}$$

En consecuencia, el coste total que el nodo *j* puede reusar del *i*, considerando ahora todas las unidades funcionales es:

$$cf_{j,i}^{tot} = \sum_{k=1}^{f} cf_{j,i}^{k,real} \qquad \textbf{[5.18]}$$

Finalmente, en un sistema formado por ambos nodos, *i* y *j*, el coste global de implementación, considerando el reuso, vendrá dado por:

$$C = c_i^{S(i)} + c_j^{S(j)} - cf_{j,i}^{tot} \qquad \textbf{[5.19]}$$

Puede parecer, a la vista de las expresiones anteriores, que el método de estimación de coste ha variado substancialmente respecto al utilizado por las técnicas previas, faltando así al principio de mínimo cambio propugnado. Sin embargo, y en contra de las apariencias, esto no es así. Quizás una reescritura de estas expresiones pueda crear un vínculo con lo explicado previamente.

La expresión [5.19] puede tomar el aspecto de:

$$C = c_i^{S(i)} + c_j^{S(j)} - \{\sum_{k=1}^{f} \hat{cf}_{j,i}^{k} \cdot (1 - \sigma_{i,j}^{k})\} \qquad \textbf{[5.20]}$$

donde para ello se ha hecho uso de las expresiones [5.17] y [5.18]. Ahora, como el coste total del nodo *j* es una constante, es posible multiplicar y dividir el sumatorio por este valor, sin que se vea modificado el resultado:

$$C = c_i^{S(i)} + c_j^{S(j)} - \frac{\{\sum_{k=1}^{f} \hat{cf}_{j,i}^{k} \cdot (1 - \sigma_{i,j}^{k})\}}{c_j^{S(j)}} \cdot c_j^{S(j)} \qquad \textbf{[5.21]}$$

Entonces, sacando factor común del coste de *j* se obtiene que:

$$C = c_i^{S(i)} + c_j^{S(j)} \cdot (1 - \{ \sum_{k=1}^{f} \frac{\hat{cf}_{j,i}^{k}}{c_j^{S(j)}} \cdot (1 - \sigma_{i,j}^{k}) \}) \qquad \textbf{[5.22]}$$

Obsérvese que este valor se ha podido introducir en el sumatorio debido a que no depende de *k*. Si ahora substituimos el último término por una variable $\rho_{j,i}$, se obtiene la siguiente expresión:

$$C = c_i^{S(i)} + c_j^{S(j)} \cdot (1 - \rho_{j,i}) \qquad \textbf{[5.23]}$$

Efectivamente, esta fórmula es idéntica a la [4.28], que representaba el coste global de dos nodos considerando el reuso. Por lo tanto, lo que parecían técnicas totalmente diferentes han resultado ser semejantes en sus expresiones. Así, para mantener la coherencia, se define el reuso vectorial del nodo *j* respecto al *i* como:

$$\rho_{j,i} = \sum_{k=1}^{f} \frac{\hat{cf}_{j,i}^{k}}{c_j^{S(j)}} \cdot (1 - \sigma_{i,j}^{k}) \qquad \textbf{[5.24]}$$

Sin embargo, los parecidos no terminan aquí. Si se observa el contenido del sumatorio anterior, se ve que viene dado por el cociente de dos costes. Más concretamente, se está representando el coste común de los nodos *i* y *j*, para una unidad funcional concreta *k*, dividido por el coste total de *j*.

Si recordamos la expresión [4.5], esta era la definición exacta de semejanza, con la única diferencia de que entonces se consideraba todo el coste reusable, y no sólo el de una unidad concreta. Por lo tanto, parece lógico establecer una nueva conexión, y definir la semejanza entre dos nodos, bajo una unidad funcional *k*:

$$\kappa_{[j,S(j)],[i,S(i)]}^{k} = \frac{\hat{cf}_{j,i}^{k}}{c_j^{S(j)}} \qquad \textbf{[5.25]}$$

De esta manera, el reuso vectorial quedaría como una simple expansión del reuso original, definido en [4.27], con la única diferencia de la aparición de un sumatorio para tratar todas y cada una de las unidades funcionales:

$$\rho_{j,i} = \sum_{k=1}^{f} \kappa^{k}_{[j,S(j)],[i,S(i)]} \cdot (1 - \sigma^{k}_{i,j}) \qquad [5.26]$$

En consecuencia, queda comprobada la total simetría entre las técnicas previas y la actual, por lo que parece claro que la expansión controlada del conjunto de variables es algo muy recomendable, con el fin de conseguir la máxima coherencia del sistema.

Una última puntualización acerca de las características de la aproximación vectorial: evidentemente, tanto la semejanza como el solapamiento de la expresión [5.26] pueden considerarse como componentes de un vector. De hecho, llamemos $\vec{K}$ y $\vec{\Sigma}$ a ambos vectores, y denotemos con $\vec{1}$ al vector identidad[7]. Entonces, se obtiene que:

$$\rho_{j,i} = \vec{K}_{[j,S(j)],[i,S(i)]} \bullet (\vec{1} - \vec{\Sigma}_{i,j}) \qquad [5.27]$$

La conclusión a la que se llega es que el reuso viene dado por la *operación del producto escalar de dos vectores, expresión que tiene aún* mayor similitud con la definición primera de este parámetro en [4.27].

5.3.3. Ejemplo de aplicación de la aproximación vectorial.

Con el fin de analizar los conceptos ofrecidos en esta sección acerca de la aproximación vectorial, se ofrecerá el modo de abordar el ejemplo presentado en este capítulo mediante el uso de dicha técnica. De esta manera, la comparación de los tres métodos de estimación explicados hasta

[7] Aquel que tiene todas sus componentes iguales a 1.

ahora (plano, escalar y vectorial) puede resultar muy clarificador, al observar *in situ* las diferencias obtenidas por cada uno de ellos.

El ejemplo estaba planteado como un caso simple de sistema, en el que únicamente se consideraban dos nodos, sin ningún tipo de dependencia. A pesar de la simplicidad de dicho caso, la posibilidad de estudiar todas y cada una de las alternativas de paralelismo existentes lo hace muy conveniente para el propósito marcado.

Así, el conjunto de pasos destinados a la aplicación de la aproximación vectorial a este ejemplo, sería el siguiente, teniendo en cuenta que los nodos escogidos son idénticos a los ofrecidos en la aproximación escalar:

- Elección del grado de paralelismo entre los nodos propuestos anteriormente como ejemplo (π=15).
- Cálculo de la expresión de los polinomios asociados a cada uno de los dos nodos. Debido a que se considera una biblioteca compuesta por nueve unidades funcionales, este será el número de polinomios existentes por nodo.
- Utilización del factor de densidad para el cálculo del solapamiento, aplicando las expresiones de la técnica vectorial explicadas en la sección anterior.
- Cálculo del coste global de ambos nodos, teniendo en cuenta el posible reuso hardware entre ellos.
- Comparación de los resultados obtenidos con los hallados mediante la técnica plana y la escalar, así como con el caso real.
- Repetición del proceso, con nuevos valores del grado de paralelismo.

Según lo expuesto, el primer paso consistiría en decidir qué puntos dentro del rango de ejecución de los nodos son considerados a fin de construir las funciones de densidad, lo cual conlleva una dificultad añadida respecto al caso previo de la técnica escalar.

Esta dificultad no sólo tiene que ver con el hecho de contar con más funciones, lo que no supondría una complejidad conceptual, sino con la base de puntos disponible para construir dichas funciones.

De esta manera, ahora se tiene que la distribución de costes está repartida entre nueve unidades distintas, por lo que el ajuste mediante una función resulta menos aconsejable.

Así, es posible que existan varias etapas de control con exactamente el mismo peso máximo de coste o distintas etapas con coste nulo, todo ello asociado a una unidad funcional en concreto.

Por lo tanto, surge la duda de qué puntos considerar para construir el polinomio. Si por ejemplo se dispone de varios puntos con un coste máximo presente, existe la posibilidad de considerar todos, lo cual llevaría a un incremento acentuado de la complejidad del polinomio. Si se opta por elegir sólo uno de ellos, existe la duda de cuál de ellos escoger. Lo mismo ocurriría al tratar el mínimo absoluto del nodo.

Al ser el objetivo de este ejemplo la comparación de resultados con los obtenidos por la técnica escalar, y al haber considerado en este caso polinomios de orden tres, con un único máximo y mínimo, es lógico que aquí también se opte por esta misma decisión, con el fin de realizar ambos de una manera similar.

En este caso, cuando existan varios puntos con valor igual al máximo *absoluto, se tomará aquel que se halle primero al recorrer el nodo* comenzando por su primera etapa de control. Lo mismo ocurrirá en el caso de varios puntos con valor igual al mínimo absoluto.

Por lo tanto, para el ejemplo presentado, y con la elección de puntos comentada, se obtiene la distribución de polinomios que aparece en la Figura 5.10.

$$\delta^1_{i,S(i)} = -1.3889\cdot x^3 + 12.5000\cdot x^2 \qquad \delta^1_{j,S(j)} = -0.0194\cdot x^3 + 0.6123\cdot x^2$$

$$\delta^2_{i,S(i)} = -0.4630\cdot x^3 + 12.5000\cdot x^2 - 100\cdot x + 250 \qquad \delta^2_{j,S(j)} = -0.0051\cdot x^3 + 0.2058\cdot x^2$$

$$\delta^3_{i,S(i)} = 2.2222\cdot x^3 - 10\cdot x^2 + 30 \qquad \delta^3_{j,S(j)} = -0.0549\cdot x^3 - 0.7407\cdot x^2$$

$$\delta^4_{i,S(i)} = 0 \qquad \delta^4_{j,S(j)} = -0.0060\cdot x^3 + 0.1333\cdot x^2$$

$$\delta^5_{i,S(i)} = -0.0174\cdot x^3 + 0.4688\cdot x^2 - 2.3438\cdot x + 3.2813 \qquad \delta^5_{j,S(j)} = -0.0086\cdot x^3 + 0.2315\cdot x^2$$

$$\delta^6_{i,S(i)} = -0.0030\cdot x^3 + 0.0667\cdot x^2 \qquad \delta^6_{j,S(j)} = 0$$

$$\delta^7_{i,S(i)} = 0 \qquad \delta^7_{j,S(j)} = -0.2222\cdot x^3 - 3\cdot x^2 + 12\cdot x - 12$$

$$\delta^8_{i,S(i)} = -0.0012\cdot x^3 + 0.0267\cdot x^2 \qquad \delta^8_{j,S(j)} = 0$$

$$\delta^9_{i,S(i)} = 0 \qquad \delta^9_{j,S(j)} = -0.1481\cdot x^3 + 0.6667\cdot x^2$$

1	2	3	4	5	6	7	8	9
*	+	-	$\leq$	$\wedge$	$\vee$	$\neg$	$\rightarrow$	$\leftarrow$

Figura 5.10

Cálculo de los vectores de densidad y unidades funcionales relacionadas

El próximo paso consiste en el cálculo de los vectores de solapamiento y semejanza, $\vec{\Sigma}$ y $\vec{K}$, según las expresiones definidas en [5.12] y [5.25]. Los resultados obtenidos son:

$$\vec{\Sigma} = (0.9772, 0.7756, 0, 0, 0.1621, 0, 0, 0, 0)$$

$$\vec{K} = (0.4500, 0.2500, 0.1000, 0, 0.0750, 0, 0, 0, 0)$$

Obsérvense dos puntos importantes tras el estudio de estos dos vectores:

- Las cuatro últimas componentes de ambos son nulas. Esto es debido a que ninguna de las unidades funcionales asociadas a ellas están presentes en los dos nodos a la vez. Por lo tanto, ni existe conflicto entre ellas, ni tampoco posibilidad de ser reusadas.
- Se obtiene un solapamiento nulo para la tercera componente, mientras que si se hace un estudio detallado, se ve que existen unidades

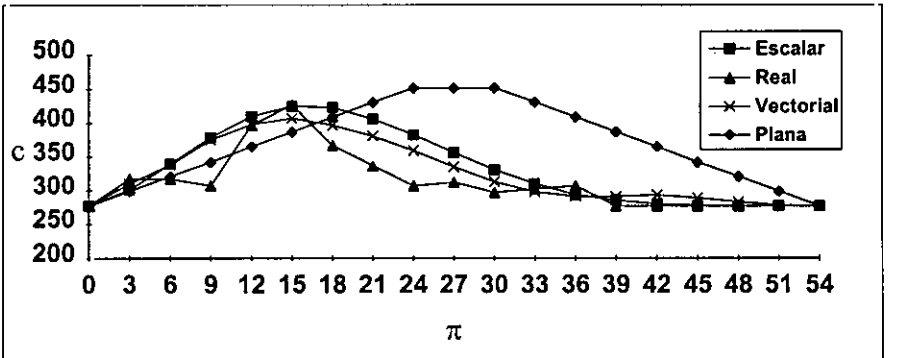


Figura 5.11

Resultados de la aproximación vectorial

funcionales comunes de ese tipo, y que además sí entran en conflicto. Este hecho se debe al error de aproximación del polinomio, lo cual es un hecho inherente a distribuciones con unidades dispersas, como suele ser el caso de la aproximación vectorial. En la sección siguiente se ofrecerá un comentario más detallado sobre este problema.

Ahora, mediante la expresión del reuso definida en [5.27], aplicada a la fórmula de cálculo del coste total [4.28], se obtiene que dicho valor equivale a 407 unidades de área. Se ve que el resultado obtenido no supera en calidad al de la aproximación escalar, que era de 425, frente al coste real, de 427.

Sin embargo, un valor aislado no es realmente significativo, por lo que es preciso realizar el estudio para todos los casos posibles de paralelismo entre ambos nodos. Los resultados obtenidos para la técnica vectorial, comparados con los de las técnicas escalar y plana, y los valores reales, se pueden examinar en la Figura 5.11.

A la vista de estas gráficas, la conclusión que se alcanza es que la tendencia de ambas técnicas es muy similar. El método vectorial no supera

de una manera definitiva al escalar, como en un primer momento se podría haber pensado.

Se observa también que los mejores resultados se obtienen en el intervalo de valores [18,30], que era donde la técnica escalar cometía un mayor error respecto a los valores reales. Se ha conseguido disminuir en parte este error, manteniendo el mismo grado de fidelidad en la curva.

El hecho de que la aproximación vectorial no supere manifiestamente a la escalar merece una serie de comentarios detallados.

5.3.4. Comentarios acerca de la aproximación vectorial.

La aproximación vectorial presentada en esta última parte del capítulo debería considerarse como una alternativa a la técnica escalar, y no como una mejora o extensión de esta. El hecho de abordarla con detalle trata de demostrar que la ampliación del conjunto de variables macroscópicas es una tarea flexible, y que puede acometerse según las necesidades propias de cada momento sólo con seguir una metodología precisa. De esta manera, y como se ha visto en la comparación del ejemplo presentado, no se puede afirmar de una forma absoluta que ninguna de ellas sea mejor que la otra.

En efecto, la aproximación vectorial, aun estando localizada dentro del nivel de trabajo macroscópico, sería algo inferior en cuanto a su grado de abstracción que la escalar. No en vano, el hecho de considerar las unidades funcionales por separado está creando una dependencia de la estructura interna del nodo, que hace que dicha técnica pierda cierto nivel de generalidad.

De esta manera, la apreciación anterior no es muy razonable en términos macroscópicos, ya que se observa claramente que la posibilidad de ajuste por polinomio no es única, y por lo tanto está sometida a mayor grado de subjetividad. Esto es debido a la dispersión de unidades funcionales, que

hace que la semántica conjunta disminuya, para dar paso a la existencia de puntos aislados.

Además, la calidad de esa aproximación no será todo lo buena que se desearía, debido a que ese nivel es proporcional al número de puntos que se traten. Al considerar el coste global de los nodos en su conjunto, se está obteniendo una visión más general de la estructura del problema. Pero cuando el número de puntos tiende a ser pequeño, la aproximación ha de sufrir forzosamente un empeoramiento.

Es lo mismo que ocurre con los métodos estadísticos, en los que el grado de fiabilidad disminuye cuando lo hace la base de muestras disponibles para realizar el estudio. Si esta base no es lo suficientemente grande, las desviaciones producidas por desajustes en las técnicas de trabajo tienen una alta repercusión, que puede llegar a ser definitiva, sobre los resultados obtenidos.

De esta manera, la técnica vectorial tendría un sentido de uso si se conoce *a priori* que las unidades funcionales que van a utilizarse poseerán una densidad de aparición apreciable, o dicho de otra manera, que la dispersión de los puntos no será demasiado influyente al pasar de trabajar con una función de densidad global, a una colección de ellas.

Aunque parezca que la técnica escalar tiene en cuenta menos detalles que la vectorial, esta afirmación no se puede considerar como totalmente cierta, a la vista de las características propias de la teoría macroscópica. De hecho, la aparente pérdida de precisión debida a la no consideración de las unidades por separado *se ve compensada por el uso del parámetro de semejanza*. En efecto, al producir un desdoblamiento de la densidad, no se está realizando un incremento de información, como *a priori* se podría pensar, sino que únicamente se está explicitando un *conocimiento ya contenido* en el parámetro de semejanza de una forma más abstracta.

De esta manera, el uso de dicho parámetro no sólo mantiene la información en el sistema, sino que además no lleva asociado el error intrínseco debido al ajuste de puntos dispersos. Este hecho es razonable si se considera que la semejanza es una magnitud muy abstracta dentro del nivel de trabajo tratado, y por lo tanto más acorde a su utilización por la teoría macroscópica.

Sin embargo, esto no quita para que la utilización de otros métodos de ajuste de funciones reduzcan drásticamente el error de aproximación, lo que haría a la técnica vectorial utilizable en más amplios espectros de ejemplos.

No en vano, se vio que la formulación que la sustenta es perfectamente compatible y deducible del resto de la teoría, lo cual le dota de los mismos principios que al resto de las técnicas, y por lo tanto, de igual grado de calidad en esencia.

De esta manera, y a la vista de las expresiones teóricas, sería totalmente imposible opinar acerca de la técnica más conveniente de utilización, escalar o vectorial, en un sentido general. El ejemplo ofrecido sencillamente muestra las tendencias de ambos métodos para un caso concreto, lo que no permite realizar ningún tipo de extrapolación.

Por lo tanto, es preciso realizar un estudio mucho más exhaustivo de todas las técnicas comentadas a lo largo de este capítulo y del anterior, comparando con rigurosidad las ventajas e inconvenientes que presentan. Para ello, en el capítulo 7, se presentará todo el trabajo experimental realizado sobre este tema, así como las conclusiones que se derivan de él.

Particionamiento.

6

Hasta ahora, he explicado las características fundamentales del entorno de Codiseño presentado, así como las técnicas de estimación de tiempo y coste asociadas al nivel de trabajo macroscópico. Sin embargo, no hay que olvidar que este proceso de estimación se desarrolla con el fin de servir como soporte al núcleo básico del Codiseño: el particionamiento hardware-software.

De esta manera, no sería lógico que en un trabajo de estas características se obviara dicho proceso. Debido a esa razón, este capítulo está destinado al particionamiento en una doble vertiente: por una parte, se comentarán las características fundamentales de los algoritmos que se han venido utilizando, todo ello de una forma genérica, y haciendo hincapié en las deficiencias que se pueden observar; por otro lado, se presentará una técnica para ser aplicada sobre los algoritmos iterativos, y que solventa ciertas dificultades asociadas tradicionalmente a este proceso.

Un punto que se debería tener en cuenta es la amplia trayectoria de investigación desarrollada sobre el campo del particionamiento. Así como los procesos de estimación han sido en parte obviados, tendiendo siempre hacia conductas simplistas, el particionamiento ha sido ampliamente estudiado y mejorado con innumerables técnicas [AlKa95] [EPKD96] [MaMo96] [Lope99] [WoJa95]. Este hecho se debe a la gran presencia de

esta tarea en otros sectores del Diseño Automático, como el particionamiento hardware-hardware, destinado a dividir un circuito entre varias pastillas de silicio, o el más moderno particionamiento temporal, ligado a los circuitos reconfigurables dinámicamente [Dave99] [MKBS99].

Con esto se quiere hacer notar el hecho de que no es posible conseguir avances espectaculares en este campo, sino que más bien se requieren modificaciones o adaptaciones de los algoritmos conocidos a problemas concretos que se presenten. Una de estas variaciones, aplicada al Codiseño, será el objetivo fundamental del presente capítulo.

6.1 Los algoritmos de particionamiento: conceptos clásicos.

Los distintos algoritmos de particionamiento existentes han sido ampliamente probados y contrastados a lo largo de muchos años de trabajo. En consecuencia, gozan de un alto grado de fiabilidad, y ofrecen garantías de consecución de buenos resultados cuando se aplican a la mayoría de los campos de investigación conocidos. Existen innumerables aproximaciones sobre las que se basan estos algoritmos.

Así, es posible encontrar tanto orientaciones centradas en metodologías matemáticas, como la *Programación Lineal Entera* [NiMa96] [NiMa97], como otras mucho más heurísticas, dedicadas a procesos *fuzzy* [LoIL98] [LoLo98].

Todas las técnicas utilizadas se pueden clasificar fundamentalmente en dos grandes grupos: los algoritmos iterativos y los constructivos. Los algoritmos iterativos tienen la particularidad de partir de una solución inicial, usualmente trivial, e ir realizando modificaciones sobre ella en pasos sucesivos, tendiendo siempre a la minimización de una función objetivo dada. Algoritmos iterativos clásicos serían el de *Fiduccia-Mattheyses*

[FiMa82] y el de *Simulated Annealing* [KiGV83]. Por otra parte, los algoritmos constructivos empiezan desde cero, y van tomando decisiones acerca de cómo construir la solución final, sin posibilidad de deshacer ninguno de estos pasos. De esta manera, tienden rápidamente hacia el objetivo final. Un ejemplo claro de este tipo de algoritmos serían los procesos de *clustering* [GDWL92].

Básicamente, el estudio realizado en el presente capítulo se centra en el perfeccionamiento de los algoritmos iterativos, debido a que suelen tener un mayor grado de precisión, a pesar de que su tiempo de convergencia pueda ser más alto que el de los constructivos.

Efectivamente, la posibilidad de los algoritmos iterativos de deshacer y repetir movimientos previamente decididos, dota a estos procesos de la posibilidad de escapar de los mínimos locales asociados a las funciones objetivo correspondientes, fenómeno este muy de desear teniendo en cuenta la abundancia de dichos mínimos, que suelen conllevar una disminución apreciable en la calidad de la solución final.

Aunque la modificación presentada en este capítulo puede ser aplicada a cualquier algoritmo iterativo, se ha elegido el de *Fiduccia-Mattheyses* para realizar el estudio conciso de los resultados.

6.1.1. El algoritmo de Fiduccia-Mattheyses.

Este algoritmo, que a partir de ahora se abreviará como FM, es uno de los más clásicos, y por lo tanto, de los más utilizados en la literatura. De esta manera, se han estudiado distintas modificaciones sobre él, con el fin de adaptarlo y mejorarlo a distintos campos de aplicación [HaHK95] [LKHC95].

Una de las ventajas que posee es su baja dependencia del problema en cuestión. Así como el *Simulated Annealing* dispone de una serie de parámetros que es necesario ajustar, según el caso que se esté tratando, el

```
Inicializar (f_mejor, Sol_mejor)
REPEAT
    Desbloquear todos los nodos
    f = f_mejor
    Sol = Sol_mejor
    REPEAT
        Estimar el nodo i, no bloqueado, cuyo movimiento produce un mejor Δf_i
        Bloquear el nodo i
        Mover el nodo i
        IF (f < f_mejor)
            f_mejor = f
            Sol_mejor = Sol
        ENDIF
    UNTIL (todos los nodos están bloqueados)
UNTIL (no se ha mejorado f en ninguna migración de los nodos)
```

Figura 6.1

Algoritmo de *Fiduccia-Mattheyses*

algoritmo de FM carece de cualquier vínculo de este tipo, por lo que es mucho más portable y versátil.

No en vano, es bien sabido que un mal ajuste del algoritmo de *Simulated Annealing*, puede producir efectos muy negativos, no sólo en el tiempo de convergencia del problema, sino también en la propia calidad de los resultados.

Una desventaja del FM, común a todos los algoritmos iterativos, es la alta complejidad asociada que tiene. En su caso más general, se puede acotar mediante $O(n^3)$, aunque modificaciones posteriores, las cuales emplean estructuras de datos más elaboradas, pueden reducir esta complejidad hasta $O(n^2 \cdot log\ n)$, que está considerada como la complejidad estándar para este tipo de algoritmos.

El algoritmo de FM se desarrolló a partir de una mejora realizada en el algoritmo de *Kernighan-Lin* [KeLi70] [BHJL89] [VaLe97], el cual a su vez

está basado en el proceso clásico de *min-cut*[1]. Su esquema, en pseudo-código, se puede observar en la Figura 6.1.

Como se puede comprobar, la alta complejidad mencionada proviene de los dos bucles anidados existentes, los cuales van explorando progresivamente el espacio de resultados, evitando los mínimos locales, hasta devolver la supuesta solución óptima. Obviamente no se puede garantizar que la solución encontrada sea la mejor posible, si para ello no se realiza un proceso de búsqueda exhaustiva, algo totalmente descartado considerando el alto tiempo de ejecución que se necesitaría.

Para progresar a través del espacio de soluciones, el algoritmo cuenta con un mecanismo de bloqueo, mediante el cual un nodo se mueve una y sólo una vez en cada paso del algoritmo. Esto es lo que se representa en el bucle interior. De esta manera, de entre todos los nodos disponibles, se evalúa cuál es más conveniente mover, a efectos de optimizar la función objetivo, *f*. Una vez tomada esta decisión, el nodo se bloquea, y se continúa con los nodos que todavía restan.

En cada uno de estos pasos intermedios, se va memorizando la mejor solución parcial encontrada. Se producirá una mejor solución parcial (*Sol_mejor*) cuando la función asociada (*f*) a una cierta solución (*Sol*), sea mejor que la hasta ahora considerada como tal (*f_mejor*).

Puede darse el caso de que tras el movimiento de un cierto nodo, cualquier cambio sobre los que todavía queden sin bloquear produzca un empeoramiento en la función objetivo. En esta situación, en la que todas las opciones serían negativas, se elegiría la que menos perjuicio provocase, y

[1] El *min-cut* es un proceso que hereda su nombre del problema de minimización del número de interconexiones que cortan la frontera entre dos circuitos. En este método están basados varios algoritmos de particionamiento, ya que, al fin y al cabo, estos también realizan tareas de optimización.

se continuaría con el algoritmo, esperando la posibilidad de una futura mejora que compense esta última decisión.

Obsérvese que el mecanismo anterior es el de escape de mínimos locales mencionado previamente. Efectivamente, el hecho de aceptar movimientos negativos hace que se tenga la capacidad de exploración más allá de la aparente solución óptima.

El bucle interior acaba cuando todos los nodos se encuentren bloqueados, ya que en esa situación, el algoritmo no puede progresar según las condiciones impuestas. Por lo tanto, se desbloquean todos los nodos, y se reinicia el proceso, en una nueva iteración del bucle exterior. Para esta nueva ejecución, se toma como partida la solución almacenada como mejor hasta ese instante.

Es evidente que este proceso ha de disponer de algún mecanismo de parada, por el cual se decida que la mejor solución encontrada hasta ese momento tiene altas posibilidades de ser la óptima. Esto se produce en el caso de que no se haya encontrado ninguna mejora de la función objetivo en una iteración completa del bucle interior. Dicho de otra manera, si partiendo de la situación en que todos los nodos están desbloqueados, se ejecuta el bucle interior, hasta que todos los nodos pasen a estar bloqueados, y en ese periodo de tiempo no se ha encontrado ninguna mejora a la función objetivo, el bucle exterior finaliza, y el algoritmo concluye definitivamente.

Como se puede comprobar, este algoritmo no ofrece garantías de encontrar la solución óptima, pero sí explora de una forma bastante exhaustiva el espacio de diseño. Es preciso realizar una nueva consideración al código presentado anteriormente. Es obvio que este código pertenece al algoritmo estándar de FM, por lo que ciertos procesos, como el de movimiento de un nodo, tenían tradicionalmente el significado de

migración de ese nodo a la partición contraria a la que estuviese asignado en ese momento.

Esta simplicidad sólo tenía sentido en los casos en los que no se consideraba la posibilidad de ejecución paralela hardware (una vez decidido que un nodo se asigne a esta partición, ¿en qué lugar se ubica? ¿cuál es el grado de paralelismo óptimo para él, lo cual tendrá influencia en el reuso?) ni de disponer de múltiples implementaciones hardware (¿con cuál de ellas se asigna a la partición?).

Es obvio que con la aceptación de los factores anteriormente mencionados, esta posibilidad de movimiento se convierte en una decisión mucho más compleja. Este problema será tratado con detalle en la sección 6.2.3.

Asimismo, se quiere hacer notar el vínculo de este proceso de particionamiento con los anteriores métodos de estimación. Se observa que dentro del bucle interior del código, se realizan una serie de decisiones críticas acerca de la conveniencia de mover un cierto nodo u otro. Para *medir esta conveniencia, es necesario comprobar la calidad de la solución* tras el movimiento, y compararla con la actual. Este proceso se realiza mediante la técnica de estimación. En otras palabras, se trata de ver cuán buena es una solución, sin necesidad de implementarla, debido al alto tiempo que esto conllevaría.

Como se puede deducir por el algoritmo y la complejidad ofrecida, un incremento, aunque parezca nimio, del tiempo destinado a la estimación, puede incidir en un aumento significativo del tiempo que consume el particionamiento. Así, la rigurosidad mostrada en capítulos anteriores acerca de la necesidad de estimadores con baja complejidad, toma plena significación ahora.

6.1.2. Limitaciones de los algoritmos iterativos: el problema de las comunicaciones.

Obviamente, y aunque los algoritmos de particionamiento han sido ampliamente estudiados y mejorados, tienen una serie de limitaciones que se manifiestan al tratar con problemas concretos.

Esto se debe a la gran versatilidad que poseen, lo que les hace ser adaptables a distintos campos de aplicación. Aparentemente, dicha versatilidad debería ser una característica favorable, y así lo es. No obstante, todos los procesos poco definidos suelen ser compatibles con un amplio espectro de problemas, pero esta generalidad es la que priva al algoritmo de la necesaria especialización requerida en tareas de alta precisión.

De esta manera, los algoritmos de particionamiento funcionan, y lo hacen correctamente, para innumerables problemas de distintos campos, pero también es cierto que podrían hacerlo mejor si se considerasen las características intrínsecas que definen a estas áreas de aplicación.

Por lo tanto, sería conveniente conjugar la probada fiabilidad de estos algoritmos, adquirida a lo largo de muchos años de trabajo, con variaciones particulares y precisas que los adapten a las circunstancias propias de cada problema, en este caso el particionamiento hardware-software.

Sin embargo, este hecho no es nada sencillo de llevar a la práctica. Así, se corre el riesgo de que la modificación introducida interfiera con el mecanismo de búsqueda del propio algoritmo, privándole de la posibilidad de encontrar la mejor solución. Por lo tanto, es necesario que las mejoras se realicen sobre problemas muy concretos de estos algoritmos, y sobre todo, que interaccionen lo mínimo posible con su estructura propia.

Una de las características importantes del proceso de Codiseño, que crea dificultades a los algoritmos de particionamiento, es el problema de las

comunicaciones. Obviamente, y como se comentó en el Capítulo 2, el hecho de trabajar con dos particiones distintas, ubicadas en lugares diferentes, hace que el intercambio de datos entre ellas consuma un tiempo considerable respecto al tiempo global de ejecución.

Este tiempo es muy superior al que necesitan los datos para transmitirse dentro de la propia partición hardware, y representa uno de los cuellos de botella del sistema. Por lo tanto, el problema de asignar los distintos nodos a sus particiones correspondientes no sólo requiere un estudio separado de cada uno de estos subsistemas, sino que es preciso comprobar la interacción entre ellos, con el fin de minimizar el tiempo de comunicaciones. Está claro que si, como ya se comentó, lo que se requiere es cumplir unas restricciones temporales sobre un sistema inicialmente implementado en software, asumiendo el perjuicio de incrementar el coste global, no sería consecuente hacerlo de tal manera que la sobrecarga por comunicaciones se dispare, ya que esta medida sería muy contra-producente.

Esto conllevaría a migrar aún más funcionalidades a hardware, aumentando el coste final, y concluyendo el proceso de Codiseño de una manera poco satisfactoria. Así, es conveniente tener en cuenta de una manera definitiva este tiempo de comunicaciones, para mantenerlo tan bajo como sea posible. Hay que recordar que este tiempo no es interesante en ningún sentido, y que si bien ha de existir, ya que se trabaja con dos particiones distintas, está compitiendo con el tiempo de operación, que es el realmente trascendental en el sistema tratado.

En este punto, y vista la importancia de las comunicaciones en Codiseño, es preciso estudiar por qué este factor crea dificultades a las técnicas de particionamiento. Todo el problema surge condicionado por una característica propia de los algoritmos estándar, que se acaba convirtiendo

en una gran limitación: la imposibilidad de mover más de un nodo simultáneamente.

Esto, que puede parecer una trivialidad, de hecho no lo es, y crea innumerables problemas cuando los tiempos de comunicación entre nodos tienen una entidad apreciable. El que los algoritmos sólo puedan mover un nodo en cada paso, no es una restricción accesoria, sino profundamente conceptual.

Así es, la exploración minuciosa del espacio de soluciones requiere de esta medida, que en caso de no cumplirse, podría distorsionar gravemente las posibilidades de alcanzar el resultado óptimo.

No sólo eso, sino que si se dotara al algoritmo de la posibilidad de movimiento múltiple, su estructura se complicaría enormemente. De esta manera, sería preciso definir cuántos nodos se podrían mover a la vez, con qué criterio se seleccionarían, cuál sería la condición de finalización, etc.

Por lo tanto, y aceptando que la posibilidad de movimiento de un único nodo es beneficiosa para la exploración del espacio de soluciones, cabe preguntarse dónde reside la problemática de este hecho con respecto a las comunicaciones.

Para comprenderlo de una manera clara, es preciso darse cuenta de la naturaleza propia de este fenómeno, el cual tiene una gran dependencia respecto al estado parcial del problema. De esta forma, la dificultad asociada a las comunicaciones reside en la imposibilidad de controlar *a priori* su peso real en el sistema, ya que este irá variando según las decisiones de particionamiento que se vayan tomando, y de igual manera, estas decisiones se verán afectadas por la situación de dichas comunicaciones.

Supongamos un ejemplo de sistema en el cual se encuentran dos nodos, con una alta carga operacional, inicialmente ubicados en la partición software. Estos dos nodos tienen una gran interacción en cuanto a datos,

por lo que la necesidad de transmisión de información entre ellos es elevada.

Como ambos se encuentran ahora en la misma partición, el tiempo de comunicación entre ellos es nulo[2]. Consideremos el caso de que, para alcanzar la mejor solución posible, dada una restricción temporal, ambos nodos deban ejecutarse en la partición hardware. Esto sería un hecho muy lógico, ya que si los dos consumen bastante tiempo en operar, lo más conveniente sería ubicarlos en hardware. Además, en este caso, la elevada comunicación de datos tampoco estaría activa, ya que los nodos se encontrarían en la misma partición.

De esta manera, se tiene el punto inicial y el punto final que el algoritmo de particionamiento debe alcanzar. Sin embargo, la situación intermedia no aparece tan clara como en un primer momento se podría suponer.

Por lo explicado anteriormente, para que los dos nodos se ubiquen en hardware, *primero es necesario mover uno, y posteriormente el otro.* Supongamos uno cualquiera de ambos nodos, y movámoslo a hardware.

En esa situación, el algoritmo debe evaluar la calidad relativa de la solución parcial, y al dispararse el tiempo de comunicación (ahora ambos nodos están en particiones distintas), probablemente será rechazada. Así, si la comunicación es muy elevada, podría enmascarar totalmente la ganancia en tiempo producida por la migración del nodo de software a hardware.

Nos encontraríamos en igual situación si en lugar de mover el nodo escogido, moviésemos el otro, ya que la comunicación se activaría de la

[2] Se considera que no existe un tiempo de comunicación definido como tal entre dos nodos en software, ya que se reserva este término para la transmisión clásica de parámetros entre el hardware y el software. El tiempo utilizado por los nodos software para leer y escribir los datos asociados se supone incluido dentro de su tiempo global de ejecución.

misma manera. Por lo tanto, se observa un estado intermedio *no estable*, cuya inestabilidad viene dada por la *aparición* de las comunicaciones.

Este hecho es el que introduce un claro factor negativo no controlable. Evidentemente, ni en el estado inicial ni en el final las comunicaciones tienen ninguna vigencia, pero en el estado intermedio repercuten tremendamente en el tiempo de ejecución del sistema.

Esto, que podría considerarse como un efecto colateral, distorsiona en gran medida el mecanismo de avance del algoritmo a través del espacio de soluciones, disminuyendo las probabilidades de encontrar la solución óptima.

Con este ejemplo se ha pretendido clarificar el problema de las comunicaciones en Codiseño, y mostrar cómo se relaciona con una característica esencial de muchos algoritmos de particionamiento: la imposibilidad de movimientos simultáneos en un único paso de ejecución.

De esta manera, y a lo largo del capítulo, propondré una técnica para solventar este efecto no deseado. Ahora, y una vez acabada esta introducción más generalista a los algoritmos de particionamiento, es preciso definir de una manera concreta cómo se engloba este proceso dentro del problema general de Codiseño, y qué relación tiene con la tarea de estimación previamente tratada.

6.2 El problema del particionamiento dentro del Codiseño.

6.2.1. Planteamiento del problema.

Es obvio que la propuesta de la técnica de estimación, explicada en el Capítulo 4 con detalle, estaba pensada para ser integrada en un mecanismo de particionamiento eficiente. Recuérdese, que entonces se comenzaba con

una solución parcial, en la que los nodos estaban asignados a una cierta partición con una implementación definida. Se suponía que esta información era proporcionada por el módulo de particionamiento y era evaluada por el de estimación.

Ahora es preciso definir claramente qué es lo que se espera del particionador, y de qué forma genera estas soluciones parciales, lo que se podría expresar de la siguiente manera:

"Dado un sistema de Codiseño inicialmente en software, representado por el grafo interno G={N,E} y la matriz de semejanza K, y una restricción temporal, t_r, hallar la función de estado S, con la correspondiente asociación de parámetros I, de tal manera que el tiempo global de ejecución cumpla la restricción impuesta con un coste asociado mínimo."

Obsérvese que lo que se pretende obtener ahora no es una solución intermedia, como se definía en el caso del proceso de estimación, sino la solución final óptima, con un tiempo de ejecución válido y un coste asociado mínimo. Implícitamente, y para comparar esta solución con todas las parciales, se está exigiendo la utilización del estimador en la definición.

Hay que recordar que cada uno de los estados intermedios generados, contaba con un cierto grado de paralelismo determinado por el uso del llamado conjunto de enlaces forzados, E_f (§4.3.2).

Este conjunto, o lo que es lo mismo, la situación relativa de los distintos nodos hardware sin ninguna dependencia común, también ha de ser proporcionada por el particionador. No aparece explícitamente incluido dentro de la definición anterior ya que forma parte directa del grafo, a través del conjunto general de aristas, E.

Los parámetros macroscópicos intrínsecos, excepto la semejanza (esto es, el tiempo, el coste y la densidad, en el caso de que se considere la estructura interna de las tareas) están implícitamente definidos dentro de cada uno de los nodos del sistema, que forman el conjunto N.

Obsérvese también que el parámetro de entrada de la restricción temporal es el factor crítico, no sólo para la solución final que se devuelva, sino también para el tiempo de cálculo global del algoritmo.

6.2.2. Nuevo concepto de movimiento asociado al proceso de estimación.

Según se comentó anteriormente, el concepto de movimiento clásico estaba ligado al simple hecho de la migración de nodos entre las dos particiones. Así, si se decidía mover un nodo que se encontrase mapeado en software, la única posibilidad existente era asignarlo a la partición hardware, y viceversa. De esta manera, este proceso de movimiento era bastante simple de abordar, pero también adolecía de las mismas carencias que los métodos de estimación tradicionales.

Como el proceso de particionamiento se va a aplicar junto con la técnica de estimación descrita en capítulos anteriores, es necesario adaptar este concepto de movimiento para que sea capaz de hacer frente a las nuevas características existentes.

De esta manera, el movimiento debe poder reflejar los conceptos descritos de paralelismo, reuso y múltiples implementaciones hardware. Para ello, se tratará de dar una nueva orientación matemática que exprese de una manera lo más rigurosa posible estas nuevas características.

A partir de ahora, un movimiento, μ, se definirá como la 4-tupla siguiente:

$$\mu = (i, e, Pred\_f_i, Suc\_f_i) \qquad \textbf{[6.1]}$$

En esta expresión, el parámetro *i* representa el nodo sobre el que se va a realizar el movimiento.

El parámetro *e* corresponde a la implementación final que este nodo va a adoptar. En el caso clásico, *e* sólo podía tomar dos valores distintos,

implementación software o hardware. Sin embargo, ahora, e puede ser cualquier elemento de las posibles implementaciones elegidas para él. Utilizando la notación del Capítulo 4, $e \in Z_i$, donde Z_i era el conjunto de implementaciones totales del nodo *i*.

Pred_f$_i$ representa el conjunto de enlaces forzados que unen los nodos predecesores de *i* con el propio *i*, y análogamente, *Suc_f*$_i$ es el conjunto de enlaces forzados que unen *i* con sus sucesores. De esta manera, se podría escribir que:

$$Pred\_f_i \cup Suc\_f_i = E_f \cap [i] \qquad \textbf{[6.2]}$$

donde [*i*] representaría el subconjunto de aristas del grafo incidentes en el nodo *i*.

Aquí aparece de una forma clara el concepto anteriormente expresado de expansión del movimiento. Ahora, ya no basta sencillamente con intercambiar los nodos entre particiones, sino que el número de posibilidades ha crecido enormemente.

Por lo tanto, y según la nueva definición, se pueden considerar tres tipos distintos de movimiento:

a) SW $\rightarrow$ HW$_i$: De software a una de las posibles implementaciones hardware.

b) HW$_i$ $\rightarrow$ SW: De hardware a software.

c) HW$_i$ $\rightarrow$ HW$_j$: De una implementación hardware a otra diferente.

La novedad de esta aproximación reside en considerar el último tipo de movimiento como tal. Esto se debe a que ahora la partición hardware es un lugar tremendamente versátil a la hora de ubicar nodos.

De esta manera, si un nodo en hardware está considerado con una cierta implementación, el particionador puede decidir que el uso de otra implementación distinta podría resultar beneficioso, de acuerdo al estado general del sistema.

Esto produciría situaciones diferentes, aunque ambas particiones estarían formadas por el mismo subconjunto de nodos, lo que conllevaría a la obtención de distintas soluciones parciales.

Pero no sólo eso, incluso si un nodo en hardware no cambia la implementación que tiene asignada, es posible que varíe su posición relativa respecto al resto de los nodos de esta partición, con lo cual de nuevo se alcanzaría una solución distinta.

Esto se debe a que no existe una ubicación prefijada para cada nodo, ya que si uno de ellos carece de cualquier tipo de dependencia respecto a otro dado, aquel podría ejecutarse en paralelo con este, pero de igual forma podría ejecutarse antes o después. Hay que recordar que el paralelismo tiene una repercusión positiva en cuanto al tiempo de ejecución, pero en igual medida la tiene negativa sobre la posibilidad de reuso, y por tanto del coste del sistema.

Así que dependiendo de las necesidades del momento se tenderá a obtener una solución más rápida y más costosa, o bien más lenta y más barata. Esto se consigue con el adecuado posicionamiento de los nodos en la partición hardware, dentro de las posibilidades existentes.

Es precisamente el particionador el que decide estas posiciones, y lo hace mediante el conjunto de enlaces forzados, E_f, que estaba representado en la definición de movimiento mediante los subconjuntos de enlaces forzados a los predecesores y sucesores del nodo que se esté tratando [MaMo97].

De esta manera, si se desea un tiempo de ejecución rápido, el conjunto de enlaces forzados será vacío, indicando que no se quieren añadir más dependencias que las naturales, y explotando al máximo el paralelismo. En cambio, si se pretende un coste reducido, el conjunto de enlaces forzados contendrá una serie de dependencias que harán que el nodo en cuestión se

Caso	E_f	t	c
(a)	∅	290	195
(b)	4→5, 5→6	300	173
(c)	6→5, 4→5	304	172
(d)	4→5, 5→2	320	169

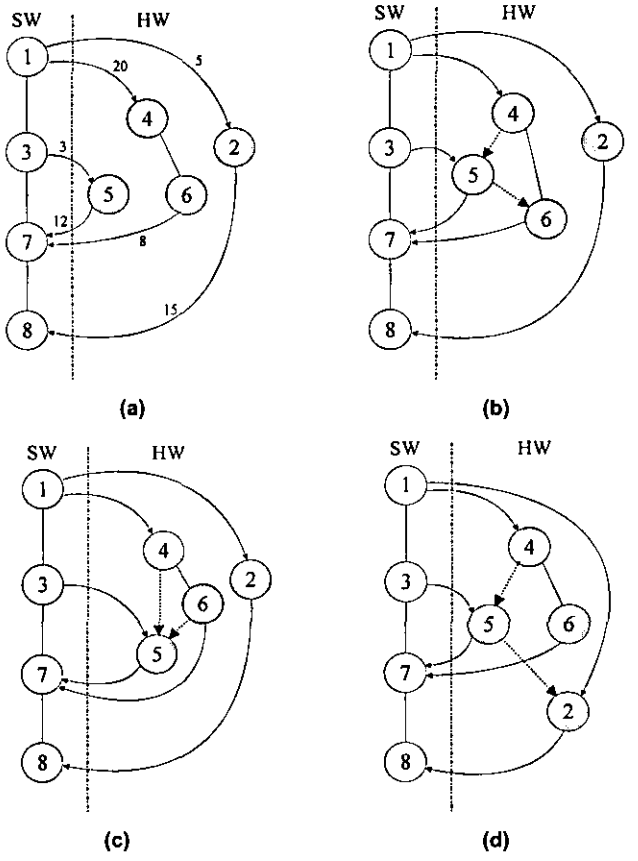


Figura 6.2

Exploración de la parte hardware

ejecute de una forma secuencial, lo que aumenta la capacidad de reuso, y disminuye el coste.

Así, se ve que la adecuada manipulación de los enlaces forzados permite la exploración de un equilibrio entre paralelismo y reuso, muy útil si

se quiere buscar en el espacio de soluciones de una forma exhaustiva. Este conjunto de enlaces debe ser decidido por el particionador, según el estado general del sistema, lo cual configurará definitivamente el conjunto de aristas totales del grafo.

Nótese aquí la riqueza de la partición hardware, que se desaprovecha totalmente por el uso de metodologías de estimación clásicas. En efecto, tradicionalmente, si se quería reducir el coste de un sistema dado, no había más remedio que mover un nodo de hardware a software. Es evidente que el objetivo propuesto se cumpliría, pero también es verdad que ese cambio probablemente produciría un incremento desmesurado del tiempo de ejecución. Esto se debe a que los nodos software son generalmente mucho más lentos que los hardware.

En cambio, ahora, este mismo resultado se puede obtener sencillamente disminuyendo el grado de paralelismo de algún nodo de la partición hardware. Esto probablemente no lleve consigo un incremento de tiempo excesivo, pero sin embargo el grado de reuso puede haber aumentado, con la consiguiente disminución del coste.

En consecuencia, se ha producido un cambio notable en las características finales del sistema, sin haber modificado la asignación de nodos a cada una de las particiones. En la Figura 6.2, se puede observar un ejemplo que pone de manifiesto este hecho. En él, un sistema de Codiseño con igual configuración de nodos en sus particiones, toma valores de área y tiempo distintos sólo con la variación del grado de paralelismo, mediante el uso de distintos enlaces forzados. Por todo lo visto, el cambio de posición de un nodo dentro de la partición hardware debe considerarse un movimiento más.

Con todas estas consideraciones, el algoritmo, una vez tenga que decidir el mejor movimiento posible en un instante dado, probará, para cada nodo software, qué ocurre al ser movido a hardware con cada una de sus

implementaciones posibles, y con cada una de sus posiciones relativas; de igual manera, estudiará para cada nodo hardware, qué ocurre al moverlo a software, y qué ocurre al cambiar su implementación y su posición relativa, de entre todas las disponibles.

Obsérvese que todo el conjunto de estas posibilidades es válido. Este estudio de movimientos realizado sobre un nodo se aplica posteriormente a todo el conjunto de ellos, lo que sin duda ofrece un amplísimo espacio de búsqueda.

Puede que el número de estas posibilidades parezca muy elevado, y de hecho lo es, pero es necesario que sea de esta manera si se quiere optar a alcanzar la mejor solución.

Es posible que ahora se tenga una visión más exacta de la complejidad del problema de Codiseño, algo que ya se explicó de una forma más intuitiva en el Capítulo 3. Por lo tanto, puede que también ahora se posea una mayor certeza de la necesidad de utilizar estimadores muy rápidos, para hacer que el proceso de particionamiento sea viable en cuanto al tiempo.

6.2.3. Adecuación del algoritmo de particionamiento al movimiento múltiple.

Una vez considerada la definición clara de la tarea que debe llevar a cabo el particionamiento, es preciso dotar al algoritmo de un mecanismo apropiado para eliminar el efecto colateral de las comunicaciones, como se comentó en la sección anterior.

Para ello, es necesario recordar el hecho aceptado de no alterar la esencia del código en cuestión, ya que es deseable que se mantenga la fiabilidad probada del proceso. Todavía queda un punto clave por tratar dentro de esta tarea, que es el interfaz o punto de unión entre el problema dado, y la actuación del algoritmo en sí.

Lo más conveniente sería *traducir* el grafo inicial del problema a otra estructura, perfectamente admisible por el particionador, de tal forma que no notara diferencia alguna respecto a cualquier otra entrada de un problema estándar, pero que sin embargo evitara de alguna manera la activación de las comunicaciones no deseables.

En otras palabras, si el algoritmo va a seguir moviendo un único nodo por turno, ya que se ha rechazado la idea de modificarlo, sería conveniente *agrupar* los nodos de la especificación inicial en entidades superiores, de tal manera que las comunicaciones costosas nunca traspasaran el límite de estas entidades [MaMC96] [MaMS98].

Así, dichas entidades han de ser isomorfas a los nodos originales de Codiseño. Por lo tanto, si tienen la misma estructura, el particionador no verá afectado su espacio de entrada, pudiendo funcionar correctamente.

Como las entidades son isomorfas a los nodos, tendrán la propiedad de indivisibilidad, y por lo tanto al mover una de estas entidades, que a partir de ahora denominaré *grupos*, se moverán *simultáneamente* todos y cada uno de los nodos asociados.

De esta manera, al realizarse el proceso de una sola vez, en ningún momento existirán nodos del mismo grupo situados en particiones distintas, y por lo tanto nunca se activarán las comunicaciones costosas entre ellos.

Gracias a esta técnica, el algoritmo sigue utilizando el mismo mecanismo tradicional, por lo que se puede contar con su fiabilidad, y el problema queda invariante, ya que los nodos originales siguen existiendo dentro de los grupos, aunque sean invisibles al particionador.

Según lo explicado hasta ahora, el proceso de agrupamiento previo podría ser considerado como una operación de *clustering* clásico. El motivo de esto es que finalmente se reduce el número de objetos que el particionador tiene que manejar, con lo que la complejidad temporal del problema se ve disminuida.

Sin embargo, existen ciertas diferencias que cambian el significado intrínseco del proceso, convirtiéndolo en algo más complicado:

a) Un hecho clave es que su relevancia, por la cual se ha escogido para trabajar con el particionador, no reside en decrementar la complejidad del proceso, sino en evitar los efectos colaterales no deseados, como ya se comentó anteriormente. En este sentido, el proceso de agrupamiento no se podría catalogar como un tipo de pre-particionamiento anterior a la ejecución del algoritmo principal, ya que no se intenta minimizar la función objetivo propuesta. En cambio, lo que se pretende podría considerarse como un cambio local en la granularidad del grafo, con el fin de facilitar la tarea del particionador. La reducción del número de entidades que se deben tratar resulta un gran beneficio en sí mismo, y ayuda a suavizar la complejidad del problema.

b) Los *grupos* formados no son estructuras planas, como los *clusters* de los procesos clásicos, sino que tienen una cierta jerarquía de nodos, con una serie de dependencias entre ellos. De hecho, los *clusters* son entidades bien definidas, sin posibilidad de cambio sobre ellos, que representan soluciones parciales dentro de un proceso de particionamiento. Sin embargo, los *grupos* podrían asemejarse a estructuras abstractas, formadas por nodos con la única característica común de ser movidos simultáneamente a través de las distintas etapas del proceso. Cuando se utiliza el concepto de *abstracto*, se quiere expresar el hecho de que los grupos no tienen una implementación definida, ni los nodos que los componen una posición fija. De esta manera, al mover un grupo a hardware es preciso *concretar* su estructura, asignando una implementación a cada uno de estos nodos, así como una posición relativa entre ellos. Esta información es preciso que sea suministrada por el particionador, ya que de lo contrario no se podrá realizar el proceso de estimación ni ningún otro de los que conforman el Codiseño. Estas

```
Inicializar (f_mejor, Sol_mejor)
Proceso de Agrupamiento
REPEAT
    Desbloquear todos los grupos
    f = f_mejor
    Sol = Sol_mejor
    REPEAT
        Estimar el grupo i, no bloqueado, cuyo movimiento produce un mejor Δf_i
        Bloquear el grupo i
        Mover el grupo i
        Mapear el grupo i
        IF (f < f_mejor)
            f_mejor = f
            Sol_mejor = Sol
        ENDIF
    UNTIL (todos los grupos están bloqueados)
UNTIL (no se ha mejorado f en ninguna migración de los grupos)
```

Figura 6.3

Adaptación del algoritmo de particionamiento

decisiones dependerán del estado del resto de los grupos del grafo, así como de las restricciones temporales. En este sentido, se toma conocimiento de que los grupos son estructuras totalmente variables, ajenas a cualquier semántica que se les pueda conceder, de no ser la del movimiento simultáneo.

De esta manera, las dos consideraciones previas clarifican el proceso de agrupamiento, tanto en su significado como en su esencia.

6.3 Extensión del algoritmo de *Fiduccia-Mattheyses*.

Una vez fijado el propósito del uso de un mecanismo de agrupamiento previo para superar las limitaciones asociadas a los algoritmos de particionamiento tradicionales, es necesario añadir las modificaciones necesarias a su interfaz para compatibilizar ambos procesos.

Hay que recordar de nuevo que estas modificaciones no han de afectar en modo alguno al funcionamiento intrínseco del algoritmo, sino que

únicamente han de servir para *traducir* el espacio de entrada y hacerlo comprensible para este.

De esta manera, sería preciso añadir dos tareas al código del algoritmo de FM presentado en la Figura 6.1. Estas dos tareas aparecen en negrita en el nuevo código de la Figura 6.3, y corresponden a los procesos de *Agrupamiento* y *Mapeado de grupos*, que explicaré en secciones sucesivas.

6.3.1. Proceso de agrupamiento.

El proceso de agrupamiento es el encargado de transformar la entrada al particionador, cambiando la estructura de nodos heredada de la especificación inicial, por otra consistente en grupos.

Evidentemente, la idea de agrupación aquí definida guarda una gran relación con el concepto de conjunto clásico. En efecto, un conjunto no es más que una asociación de elementos con alguna propiedad en común, que permiten un cierto trato generalista, pero que siguen manteniendo su esencia individual.

Esto se puede extrapolar al presente caso, en el que los grupos corresponderían a conjuntos de nodos cuya característica común sería el movimiento simultáneo a través de las distintas particiones.

Sin embargo, todo conjunto tiene intrínsecamente asociada una condición de pertenencia, que lo representa de una forma unívoca. En otras palabras, *para formar un conjunto es preciso delimitar qué condiciones* deben cumplir los elementos del universo para pasar a considerarse miembros. Es lo que en términos formales se denomina una *función de pertenencia*.

Por lo tanto, este hecho también es aplicable al concepto de grupo, y es preciso determinar, mediante una relación matemática, qué condición deben cumplir los nodos para formar parte de un cierto grupo. Este criterio es lo que se conoce como *criterio de agrupamiento*.

Antes de continuar es preciso dejar claro un hecho muy importante relacionado con el agrupamiento. Si se decide que un determinado número de nodos se ha de mover simultáneamente, esto conlleva lógicamente el hecho de que todos ellos han de estar asignados a idéntica partición en todo momento.

Aunque esta afirmación pueda parecer trivial, encierra un efecto colateral muchas veces no deseado. Así, se está privando al particionador de la posibilidad de explorar aquellas soluciones en las que los nodos de un mismo grupo están asignados a particiones distintas. Si una de estas soluciones es la óptima, será totalmente imposible alcanzarla.

En este punto, la conclusión a la que se debe llegar es que el proceso de agrupamiento es beneficioso para escapar de ciertos mínimos locales que presente el espacio de soluciones, pero por otra parte, si no se elige convenientemente, puede enmascarar parte de ese espacio de soluciones, haciendo imposible alcanzar el resultado óptimo.

Por lo tanto, y partiendo del hecho de que debe existir un criterio de agrupamiento, el siguiente punto por tratar sería la determinación de dicho criterio. No obstante, hay que decir que debido a los múltiples factores internos y externos al sistema que tienen influencia en este problema, no existe una función absoluta que determine el mejor criterio de agrupamiento para todos los casos.

Es decir, no existe ninguna relación matemática que dado el problema inicial, sea capaz de predecir el mejor criterio para todos y cada uno de los casos posibles de restricción temporal que se puedan imponer.

Por esta razón, el criterio de agrupamiento debe ser heurístico, basado en la experiencia previa, y que ponga de manifiesto aspectos concretos y conocidos del problema tratado.

De esta manera, y teniendo en cuenta la dificultad explicada, el siguiente objetivo es proponer una relación válida que permita optar por, si

no se puede garantizar el mejor[3], un adecuado criterio de agrupamiento para cada situación propuesta. Quizás, la deducción que viene a continuación pueda parecer un poco artificiosa y generada de una forma poco natural. Sin embargo, es consecuencia de un análisis amplio de problemas de particionamiento en Codiseño.

Así, en lugar de realizar una deducción *a priori*, proponiendo un modelo teórico basado en formulaciones anteriores, y probándolo experimentalmente después, se ha optado por una deducción *a posteriori*, en el que el modelo ha sido ajustado progresivamente a medida que se han realizado pruebas experimentales, concluyendo en las expresiones matemáticas que se comentarán a continuación.

Por lo tanto, los siguientes pasos están destinados a proponer una relación válida que permita hallar el criterio de agrupamiento adecuado, τ.

a) Valor de la comunicación.

El primer parámetro que puede resultar influyente en este criterio, es el valor absoluto del tiempo asociado a cada comunicación c, y que representaré por $o(c)$.

Es obvio que este factor ha de resultar determinante, ya que lo que se intenta calibrar es si una comunicación resulta excesiva o no. Así, y aunque esta decisión sea subjetiva, el tiempo absoluto asociado a cada una de estas comunicaciones resulta esencial.

b) Restricción temporal.

Otro factor que sin duda tiene una gran influencia es la restricción temporal impuesta, t_r. Al comenzar siempre en una solución trivial con todos

[3] Véase la sección 6.3.3 para solventar la problemática de no poder garantizar el criterio de agrupamiento óptimo.

los nodos en software, pueden darse dos casos extremos bien diferenciados: que la restricción temporal sea muy relajada, con lo que bastará que pocos nodos sean migrados a la partición hardware (y por lo tanto se necesitarán pocos movimientos del particionador); o que la restricción temporal sea muy estricta, con lo que será más difícil de cumplir, y por lo tanto se precisará mover más nodos a hardware.

De esta manera, en el primer caso se podría considerar que la solución óptima tiene una posición muy superficial dentro del espacio de búsqueda, o en otras palabras, que se encuentra muy próxima al estado inicial del problema. En cambio, en el segundo caso la solución óptima se encuentra en una posición más profunda, y el particionador debe realizar más iteraciones para llegar hasta ella, así como más movimientos de nodos.

En resumen, cuando la restricción temporal es relajada, se necesitan pocos movimientos de nodos, con lo que se producirán menos activaciones de comunicaciones, y el valor absoluto de estas tendrá menos relevancia. Sin embargo, si la restricción temporal es estricta, el número de movimientos de nodos será mucho mayor, produciéndose un gran número de activaciones en las comunicaciones, con lo que su valor será determinante. Por lo tanto, la restricción temporal matiza el valor absoluto asociado a las comunicaciones del sistema.

c) Probabilidad de activación de una comunicación.

Un tercer factor es la probabilidad que tiene una comunicación, *c*, de ser activada, $\pi(c)$. Es evidente que si una comunicación no va a activarse, no tiene ninguna importancia cuál es su valor absoluto, $o(c)$, ya que nunca influirá en el tiempo de ejecución global del sistema. Ahora bien, el que una comunicación se active o no, depende de si sus nodos adyacentes serán movidos en el proceso.

De esta manera, si denotamos por $\tilde{\pi}(i)$ a la probabilidad normalizada[4] de mover el nodo *i*, y se supone que *i* y *j* son los nodos adyacentes a la comunicación *c*, se tiene que:

$$\pi(c) = \frac{[\tilde{\pi}(i) + \tilde{\pi}(j)]}{2} \qquad \textbf{[6.3]}$$

Un hecho importante es que al comenzar ambos nodos en software, basta con que se decida mover uno de los dos, o ambos, para que la comunicación se active. Puede llevar a error el hecho de pensar que si ambos nodos tienen una probabilidad muy alta de ser movidos a hardware, entonces la comunicación estará inactiva, ya que los dos compartirían la misma partición. Esto no es así, porque como se explicó anteriormente, es en el instante intermedio en que un nodo ya se ha movido y el otro todavía no, cuando se activa la comunicación, evitando la posibilidad de completar el movimiento.

De la explicación anterior se deduce que $\pi(c)$ ha de ser directamente proporcional a $\tilde{\pi}(i)$ y $\tilde{\pi}(j)$. La razón de dividir ambas probabilidades por 2 es debido al hecho de mantener el resultado en el rango [0,1], lo cual equivale a un simple cambio de escala.

Falta dar una expresión válida a la probabilidad $\tilde{\pi}(i)$ de mover un nodo *i* a la partición hardware. Para ello, es necesario comprender las distintas políticas de movimiento que posee el particionador, con el fin de hacer frente a los variados estados en los que el sistema se puede encontrar.

Si el tiempo de ejecución del sistema está lejos de cumplir la restricción temporal impuesta, entonces predomina una fuerte política de reducción de este tiempo, aun incrementando de una manera excesiva el coste. Recuérdese que el cumplimiento de la restricción temporal es el objetivo prioritario del proceso.

[4] Concepto que se explicará más adelante en esta misma sección.

Sin embargo, si el tiempo de ejecución está próximo a la restricción, lo que prevalece es una política de optimización del coste, que es el objetivo secundario propuesto.

De esta manera, al principio del proceso se tenderá a usar la primera política, y se irá cambiando a la segunda a medida que el particionador se *acerca a la solución óptima. Está claro que cada movimiento de un cierto* nodo producirá un efecto determinado sobre el coste y el tiempo del sistema. Así, dependiendo del estado general en que se encuentre el problema, será más conveniente mover un nodo u otro, y por lo tanto se aumentarán o disminuirán sus probabilidades de movimiento.

En consecuencia, hallar la probabilidad de estos nodos es una tarea compleja, sobre la que influyen factores internos y externos al propio nodo. Para calibrar este factor, definamos primero la ganancia de tiempo y coste al mover un nodo *i* a hardware, que denominaré $r_t(i)$ y $r_c(i)$ respectivamente.

Las expresiones matemáticas asociadas vendrían dadas por:

$$r_t(i) = t_i^{SW} - \frac{\sum_{j=1}^{m(i)} t_i^{HW_j}}{m(i)} \qquad \textbf{[6.4]}$$

$$r_c(i) = \frac{\sum_{j=1}^{m(i)} c_i^{HW_j}}{m(i)} - c_i^{SW} \qquad \textbf{[6.5]}$$

Recuérdese que *m(i)* representa el número total de implementaciones hardware escogidas para el nodo *i*. Es obvio que *a priori*, cuando se está realizando el proceso de agrupamiento, no se puede conocer la implementación final que el particionador va a elegir para un nodo determinado, y por lo tanto no es posible calcular las ganancias de tiempo y coste exactas en ese caso.

Así, se puede observar que los factores propuestos consideran una media aritmética de las distintas implementaciones hardware respecto al caso de asignación software. Obsérvese también el orden en el que aparecen los términos en ambos factores, escogido así para garantizar la consecución de valores positivos[5].

De esta manera, la probabilidad (no normalizada) de mover un nodo a hardware será tanto más favorable cuanto mayor sea su decremento de tiempo, y tanto más desfavorable cuanto mayor sea su incremento en coste. Así, se puede expresar matemáticamente de la siguiente forma:

$$\pi(i) = \frac{f_t \cdot r_t(i)}{Tmax} - \frac{f_c \cdot r_c(i)}{Cmax} \qquad \textbf{[6.6]}$$

Los valores de *Tmax* y *Cmax* corresponden, respectivamente, al tiempo máximo que puede alcanzar el circuito (equivalente al caso de que todos los nodos se encuentren mapeados en software) y al coste máximo asociado (que ocurrirá cuando todos los nodos estén mapeados en hardware con su implementación más rápida).

De esta forma, las ganancias de tiempo y coste, $r_t(i)$ y $r_c(i)$, se encuentran normalizadas, ya que por fuerza deben ser menores que estos valores.

Falta por explicar el significado de las funciones f_t y f_c, que aparecen en el numerador de la expresión. Estas funciones son las que precisamente determinan qué tipo de política se va a llevar a cabo, si la optimización de tiempo o de coste.

---

[5] Este orden se debe a que el tiempo de ejecución software es mayor que cualquier tiempo de ejecución hardware, y por lo tanto también es mayor que su media. Justamente lo contrario ocurre en el caso del coste, que se considera nulo para el software y positivo para cualquier implementación hardware.

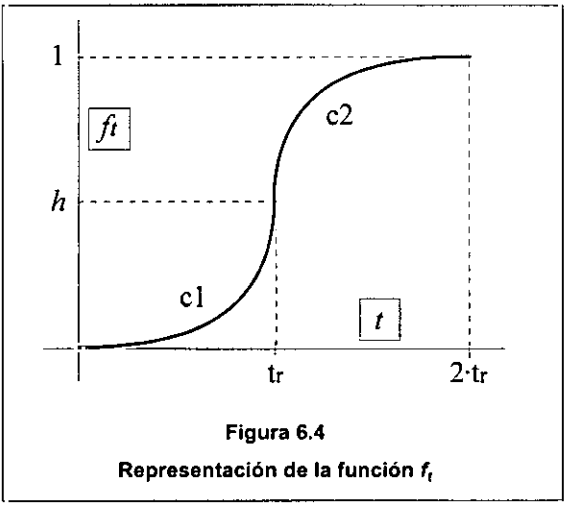


Figura 6.4
Representación de la función f_t

Si es preciso disminuir mucho el tiempo, debido a que la restricción temporal todavía queda lejos, f_t será mayor que f_c, con lo que se potenciaría el primer término de la expresión, y por lo tanto la búsqueda de movimientos que sean capaces de reducir este tiempo. Justamente ocurriría lo contrario si la restricción temporal ya ha sido cumplida: se primaría el segundo término y el movimiento de nodos encaminados a disminuir el coste del sistema.

La formulación de estas dos funciones debe ser tal que represente de una manera clara la semántica previa. Así, es posible encontrar distintas expresiones, todas ellas correctas, que tendrán que ser ajustadas experimentalmente con el fin de conseguir los mejores resultados posibles.

En la Figura 6.4 se puede observar una expresión para f_t que ha sido calibrada, y que está construida mediante aproximación exponencial. Este tipo de ajuste es muy conveniente para todos los casos en los que se pretenda una variación suave y progresiva del parámetro que se está intentando modelar.

Se observa que existen dos zonas diferenciadas, aquella donde el tiempo de ejecución es menor que la restricción temporal (y por lo tanto se cumple), en la que la función disminuye mucho, indicando que no interesa la política de reducción de tiempos; y aquella en la que el tiempo es mayor que la restricción temporal (la cual no se cumple), donde la función crece e indica que sí conviene reducir el tiempo del sistema.

Claramente, esta función pone de manifiesto las ideas expuestas anteriormente. Destacar que todos sus valores están comprendidos en el intervalo [0,1]. Además, el rango de su dominio está restringido al intervalo de tiempos $[0,2 \cdot t_r]$. En el caso de contar con valores superiores a este intervalo, se les asigna la ordenada del punto $2 \cdot t_r$, debido a que al estar ya muy alejados de la restricción temporal, se impone una política máxima de reducción de tiempo. La expresión matemática de esta función f_t vendría dada por la ecuación:

$$f_t(t) = \begin{cases} f_1, \text{ si } t \le t_r \\ f_2, \text{ si } t_r < t \le 2 \cdot t_r \\ 1, \text{ si } t > 2 \cdot t_r \end{cases} \qquad \textbf{[6.7]}$$

$$f_1 = c_1 \cdot (e^{\frac{t}{t_r} \cdot log(1+\frac{h}{c_1})} - 1) \qquad \textbf{[6.8]}$$

$$f_2 = \{(c_2+1) \cdot (1 - e^{\frac{(t-t_r)}{t_r} \cdot log\frac{(c_2+h)}{(c_2+1)}})\} + h \qquad \textbf{[6.9]}$$

Se observa en las expresiones que existen una serie de parámetros que sirven para calibrar o dar forma a la función. Estos parámetros son c_1, c_2 y h. Los dos primeros están orientados a definir la curvatura de cada una de las dos ramas de la función. Un valor de 1 para ambas sería algo habitual,

pero se deja libertad al diseñador para que modifique esta curvatura de una manera detallada.

El parámetro h sirve para marcar la altura de la gráfica en el punto de intersección de ambas ramas, justo donde la abscisa toma el valor de t_r. Si h toma un valor alto ($h \in [0.5, 1]$), significa que aun cuando se haya cumplido la restricción temporal, se seguirá efectuando una política de reducción del tiempo exigente. En cambio, si h toma un valor bajo ($h \in [0, 0.5]$), indica que aun no habiéndose cumplido la restricción, la política de reducción del tiempo se ve relajada.

Es conveniente que este valor tienda a ser alto, ya que cualquier optimización del coste lleva consigo un incremento del tiempo. Así, puede darse el caso de una situación en la que habiendo cumplido la restricción impuesta, se intente disminuir el coste, lo que provocaría un incremento del tiempo, incumpliéndose de nuevo la restricción, y dando un paso atrás en el proceso de particionamiento.

Se puede comprobar que las expresiones [6.8] y [6.9] equivalen a las dos ramas de la función propuesta (Figura 6.4), mediante el cálculo de los límites por los extremos de ambas:

$$\begin{cases} \lim_{t \to 0} (f_1) \to 0 \\ \lim_{t \to t_r} (f_1) \to h \end{cases} \qquad \textbf{[6.10]}$$

$$\begin{cases} \lim_{t \to t_r} (f_2) \to h \\ \lim_{t \to 2 \cdot t_r} (f_2) \to 1 \end{cases} \qquad \textbf{[6.11]}$$

Aparte de esto, sería necesario definir la función f_c. No se detalla su expresión, ya que se ha elegido de tal manera que $f_c = 1 - f_t$.

Con todas estas definiciones, la expresión [6.6] queda completa. Según los rangos de valores presentados para cada uno de los términos, se cumple que $\pi(i)$ debe estar comprendida en el intervalo [-1,1].

Este rango sirve de indicación de la *fuerza* con la que se está intentando optimizar los parámetros del sistema. Cuando $\pi(i)$ tienda a -1, *significará que se trata de reducir con mucha intensidad el coste del* sistema, y por lo tanto, es muy poco probable que el nodo *i* se mueva a hardware. Por el contrario, si $\pi(i)$ tiende a 1, lo que representa es que se intenta minimizar el tiempo del sistema, con lo que es bastante probable que el nodo *i* se mueva a hardware. En situaciones intermedias, $\pi(i)$ adoptará un valor próximo a 0.

Sin embargo, y como la probabilidad de movimiento del nodo ha de servir para calcular la probabilidad de activación de la comunicación adyacente, es más adecuado que su rango de valores esté comprendido en el intervalo [0,1]. Así, cuando esta probabilidad se acerque a 0, indicará que el nodo tiende a quedarse en la partición software, mientras que cuando tienda a 1 representará un casi seguro movimiento a hardware. De esta manera, se define la probabilidad normalizada de movimiento de un nodo *i*, como:

$$\hat{\pi}(i) = \frac{\pi(i)+1}{2} \qquad \textbf{[6.12]}$$

Está claro que el nuevo valor así definido está comprendido en el intervalo [0,1].

d) Número de comunicaciones activas.

El último factor que se considerará a la hora de hallar el criterio de agrupamiento es el número potencial de comunicaciones activas en el sistema. Es obvio, que cuanto mayor sea el número de comunicaciones

activas, más alto será el tiempo empleado en realizarlas, y por lo tanto, la influencia negativa en el tiempo global de ejecución también será más elevada.

De esta manera, si se tiene una restricción impuesta fija, que por condición del problema no se puede violar, ese tiempo limitado del que se dispone se ha de repartir entre el tiempo de operación de los nodos y el tiempo de comunicación. Está claro que lo que realmente interesa es maximizar el primer factor, ya que si el segundo se dispara en exceso, se estaría empobreciendo el rendimiento del sistema.

En consecuencia, y teniendo en cuenta que el tiempo de comunicación se quiere mantener bajo unos mínimos razonables, si va a existir un número alto de estas comunicaciones activas en el sistema se deberá ser muy estricto con la sobrecarga que producen, ya que la suma de todas ellas podría ofrecer un total inadmisible. En cambio, si el número de comunicaciones activas va a ser muy bajo, se puede ser más permisivo, ya que es probable que su baja presencia no altere demasiado el esquema de tiempos del sistema.

Por lo tanto, y para tener en cuenta este hecho dentro del modelo de agrupamiento, se considera el denominado *factor de corrección* γ. A la hora de dar una expresión asociada a este factor, es preciso comprender cómo evoluciona la presencia de comunicaciones activas en el sistema, según las condiciones fijadas.

Si la restricción impuesta, t_r, es muy rigurosa, o por el contrario, muy relajada, la mayoría de los nodos tenderán a ubicarse en la partición hardware o en la partición software, respectivamente. Esto quiere decir que en cualquiera de los dos casos existirá una partición dominante, en la que se desarrollarán la mayoría de las operaciones, mientras que la partición opuesta tendrá una actividad esporádica.

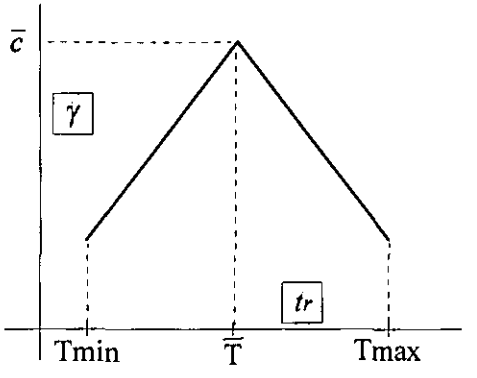


Figura 6.5

Representación de la función γ

Como consecuencia, las comunicaciones entre ambas particiones se producirán de una manera poco frecuente, o en otras palabras, el número de comunicaciones activas será bajo.

Sin embargo, puede darse el caso de que la restricción impuesta se corresponda a un punto temporal centrado entre los dos extremos anteriores. Entonces, las dos particiones tenderán a ser más equilibradas, repartiéndose las funcionalidades entre ellas de una forma no polarizada. En ese caso, cuando ambas particiones funcionen de una manera continuada, es cuando la carga de comunicaciones será más intensa entre ellas, lo que equivale a decir que el número de comunicaciones activas será más alto.

De esta manera, es preciso caracterizar al factor γ con una expresión matemática que represente la semántica explicada. Al igual que ocurría con los factores f_t y f_c, existen distintas posibilidades que pueden funcionar correctamente en este caso.

Se ha elegido una función en punta de flecha, la cual se puede observar en la Figura 6.5, debido a su simplicidad en el manejo. Obviamente, esto indica que se considera una variación lineal del número

de comunicaciones activas en el sistema. También se ve claramente que esta función no es de clase C^1, debido a la existencia del punto central, y por lo tanto, ofrece una variación menos suave que la gráfica exponencial presentada para f_t.

En aquel caso sí que era muy conveniente esa variación no brusca, debido a que existían dos factores conjugados, de tal forma que el cambio en un sentido de uno producía el mismo efecto, de sentido contrario, en el otro ($f_c = 1 - f_t$). Sin embargo, en este caso, la variación lineal de γ será suficiente para introducir el efecto de las comunicaciones en el sistema.

La expresión matemática de este factor de corrección viene dada por:

$$\gamma(t_r) = \begin{cases} \gamma_1, & \text{si } t_r < \overline{T} \\ \gamma_2, & \text{si } t_r \geq \overline{T} \end{cases} \qquad \textbf{[6.13]}$$

$$\gamma_1 = \frac{\overline{c}}{\overline{T} - Tmin} \cdot (t_r - Tmin) \qquad \textbf{[6.14]}$$

$$\gamma_2 = \frac{\overline{c}}{\overline{T} - Tmax} \cdot (t_r - Tmax) \qquad \textbf{[6.15]}$$

Los valores de *Tmin* y *Tmax* corresponden a los de menor y mayor tiempo que el circuito puede alcanzar. Estos estarían asociados al caso en el que todos los nodos estuviesen asignados a hardware, con su implementación más rápida, y al caso en el que todos los nodos estuviesen en software, respectivamente.

El valor de $\overline{T}$ corresponde al punto medio aritmético entre *Tmax* y *Tmin*, el cual se ha considerado como el caso de máxima comunicación entre las dos particiones. Siguiendo con esta argumentación, se asigna al anterior punto el valor de $\overline{c}$, como el máximo de esas supuestas

comunicaciones activas, y que representa el valor medio de las comunicaciones totales existentes en el sistema[6].

Observar que la función γ es continua, ya que se cumplen los valores extremos:

$$\begin{cases} \lim_{t_r \to Tmin} (\gamma_1) \to 0 \\ \lim_{t_l \to \overline{T}} (\gamma_1) \to \overline{c} \end{cases} \quad [6.16]$$

$$\begin{cases} \lim_{t_r \to \overline{T}} (\gamma_2) \to \overline{c} \\ \lim_{t_r \to Tmax} (\gamma_2) \to 0 \end{cases} \quad [6.17]$$

Una vez llegados a este punto, todos los parámetros que intervienen en el modelo de agrupamiento han sido explicados, y sólo falta conjugarlo adecuadamente para así construir una formulación que los relacione. De esta manera, se propone la siguiente expresión:

$$\{ \sigma(c) \cdot \gamma \cdot [n \cdot \pi(c)] \} > t_n \quad \forall c \quad [6.18]$$

El factor n representa al número total de nodos en el sistema, y aparece en la expresión para que el orden de magnitud de ambos miembros sea equivalente. Obsérvese que esta relación, aplicable a todas las comunicaciones del sistema, será tanto más fácil de cumplir cuanto:

- Mayor valor absoluto tenga la comunicación.
- Mayor sea el número de comunicaciones activas (γ).
- Mayor sea la probabilidad de activar esa comunicación.

[6] Es razonable considerar que si ambas particiones están equilibradas, todos los nodos tendrán un número semejante de comunicaciones con aquellos de su misma partición (desactivadas) que con los de la partición opuesta (activadas). Siempre, claro está, hablando en término genéricos.

- Más estricta sea la restricción temporal.

De esta manera, se puede calibrar de una manera más objetiva si una comunicación es suficientemente costosa como para tener que ser agrupada o no.

A partir de aquí, es preciso determinar el mecanismo para el cálculo del criterio de agrupamiento, τ, que era el objetivo prioritario de esta sección. Para ello, el algoritmo propuesto es el siguiente:

- Ordenación de las comunicaciones existentes en el sistema, según el tiempo que consuman.
- Recorrido de la lista ordenada de comunicaciones, empezando por aquella que tenga menor tiempo asociado, y aplicando a cada uno de los elementos la relación [6.18].
- Detener el proceso en el momento en que se encuentre la primera comunicación que cumpla dicha relación. Entonces, se considera esa comunicación como merecedora de ser agrupada, y pasa a determinar el criterio de agrupamiento, τ.

De esta manera, y antes de comenzar el proceso de particionamiento, todos los nodos con comunicaciones adyacentes superiores al valor calculado se colapsan en grupos mediante un proceso asociativo.

Al finalizar, se tendrá un nuevo espacio de entrada, con los grupos sustituyendo a los nodos originales como entidades de particionamiento, y donde todas las comunicaciones visibles existentes son menores que el criterio de agrupamiento calculado.

Hay que decir que la relación presentada anteriormente puede parecer deducida de una manera poco natural. Asimismo, cabe preguntarse el porqué de la consideración de los parámetros explicados, y no de otros, también razonables.

El motivo de esto ya fue comentado al principio de esta sección, pero es preciso reiterar que el proceso que ha llevado a las conclusiones

referidas es puramente deductivo, o *a posteriori*. Así, pueden existir innumerables criterios para determinar el método de agrupamiento, todos ellos válidos. Lo que es seguro es que el aquí presentado se ha elaborado tras el estudio y prueba de ejemplos de particionamiento, dando buenos resultados, como posteriormente se comprobará.

El hecho de abordar así el problema es bastante lógico, ya que nos movemos en el campo de la heurística, terreno este totalmente relacionado con la experimentación y calibrado de parámetros. Por supuesto, la fase experimental desarrollada es mucha más amplia que la ofrecida al final de este capítulo, donde sólo se muestra un ejemplo explicativo. Véase el capítulo 7 para un estudio detallado de los resultados obtenidos tras la aplicación de esta técnica.

6.3.2. Proceso de mapeado de grupos.

Una vez determinado el criterio de agrupamiento, y por lo tanto los grupos conformados a partir de la especificación inicial, el proceso de particionamiento puede dar comienzo. Así, se podrá realizar con el algoritmo estándar seleccionado, ya que no encontrará diferencias entre la circunstancia actual y la previa de no agrupamiento, donde los nodos originales eran entidades visibles.

De esta manera, y una vez comenzado el proceso, surge una nueva dificultad anteriormente inexistente, y que está relacionada con las características intrínsecas de los grupos. Esto es debido a que estas entidades, como ya se comentó, son abstractas, en el sentido de que no están vinculadas a ninguna implementación real. En consecuencia, y si la única relación en común de los nodos integrantes es su movimiento simultáneo a través de las distintas etapas, es preciso indicar una metodología adecuada a la hora de ser implementados.

En esta situación, si un grupo va a ser asignado a la partición software, no existe ningún problema adicional, ya que se está considerando la presencia de un único procesador, y el orden de ejecución de los nodos ya está prefijado por el compilador del sistema, teniendo en cuenta las posibles dependencias entre ellos.

Sin embargo, si el grupo ha de ser asignado a hardware, debido a las múltiples posibilidades comentadas a lo largo del capítulo, surge una indeterminación acerca de las distintas relaciones reales de los nodos que lo componen y su implementación.

De esta manera, si suponemos un grupo formado por una serie de nodos totalmente independientes, y si se decide su ubicación en la partición hardware, es posible situar a todos ellos en paralelo, pero de igual manera se puede optar por una ejecución secuencial.

Este es el mismo problema que se presentaba al mover un único nodo, en el que era preciso determinar su relación con los otros situados en la misma partición. No obstante, esta cuestión aparece ahora acentuada, ya que no sólo es preciso calcular estas características, sino también hay que decidir la relación entre los distintos nodos dentro del grupo.

Para solventar esto, existen dos posibles alternativas:

a) Calcular todos y cada uno de los diseños válidos para la estructura del grupo que se vaya a mover, respecto a todas las posibles situaciones relativas dentro de la partición hardware, considerando para ello dos aspectos. El primero está relacionado con la implementación de cada nodo del grupo, así como con los posible enlaces forzados entre ellos; el segundo aspecto está vinculado a los posibles enlaces forzados de los nodos del grupo respecto al resto de nodos de la partición. Evidentemente, las posibilidades totales existentes para un único movimiento son inmensas, y ralentizarían mucho el proceso total de particionamiento. Por lo tanto, no es posible explorar todas las

alternativas, siendo necesario la utilización de algún método heurístico que simplifique el tiempo consumido por cada movimiento.

b) Calcular la implementación y posición relativa dentro de la partición hardware de cada nodo individualmente, eligiéndolos uno a uno de entre todos los que conforman el grupo. De esta manera, se escogería el primer nodo del grupo y se implementaría; después se repetiría el proceso con el segundo nodo, teniendo en cuenta que el estado general del sistema ha cambiado debido a las decisiones previas, y así sucesivamente hasta que se acabe con todos los nodos del grupo. En cada punto, es obvio que hay que tender hacia elecciones que mejoren la función objetivo, aunque la imposibilidad de dar marcha atrás en los nodos implementados, hace que exista la posibilidad de caer en mínimos locales, si no se realiza el proceso cuidadosamente.

Se ha optado por la segunda opción debido al tiempo considerablemente menor que requiere, aunque introduzca una mayor dificultad conceptual. Esta dificultad viene motivada por el hecho de que las decisiones que se tomen en las primeras etapas, pueden cambiar las circunstancias del problema de tal manera que influyan determinantemente en las últimas alternativas, incluso conduciendo al proceso hacia zonas de soluciones no deseadas.

En consecuencia, el orden de asignación de los nodos a la partición hardware es vital, y por lo tanto la construcción de la lista de prioridad que ordene dichos nodos es crítica.

El hecho de que se hable de asignación ordenada de nodos, no significa en absoluto que se haya perdido el concepto de simultaneidad. En efecto, aunque estas asignaciones se realicen de una en una, el movimiento del grupo sigue siendo único. Esto es porque entre cada una de las asignaciones de nodos no se estima la solución resultante, y por lo tanto no se puede rechazar hasta que la totalidad de estos nodos hayan sido

movidos. En este sentido, desde el punto de vista del particionador, el movimiento del grupo sigue siendo monolítico.

Con esta consideración, el movimiento de un grupo *g* corresponde a la aplicación de un conjunto de movimientos individuales sobre nodos, $M_g = \{\mu_k\}$, donde *k* representa un índice que recorre la lista de prioridad asociada a ese grupo.

Asumamos que el grupo *g* está formado por los nodos {*a,b,...,z*}. De esta manera, y por lo comentado anteriormente, las decisiones tomadas acerca del movimiento del nodo *i*, μ_i, se verán afectadas por todas las decisiones tomadas previamente, para los movimientos, $\mu_a,...,\mu_{i-1}$, y a su vez afectarán a los movimientos subsiguientes, $\mu_{i+1},...,\mu_z$.

A la hora de establecer la lista de prioridad, es importante que responda a dos factores propios del problema: los parámetros internos de cada nodo, y el estado general del proceso de particionamiento.

De esta manera, si el tiempo del sistema no cumple las restricciones impuestas, sería deseable mover *primero* a hardware aquellos nodos dentro del grupo que produjesen un mayor decremento de este tiempo. Análogamente, si la restricción ya casi ha sido cumplida, los mejores candidatos para ser asignados en primer lugar a hardware serían aquellos nodos que no introdujesen un excesivo incremento en el coste.

Sin embargo, si se observan las condiciones comentadas en el párrafo anterior, se ve que son idénticas a aquellas presentadas en la sección 6.3.1, y que servían para definir las ganancias de tiempo y coste usadas en el cálculo de la probabilidad de movimiento de un nodo.

Se comprueba que la utilización de dicha probabilidad, π, sería un factor adecuado para la ordenación de la lista de nodos. Así, se moverían aquellos con una mejor ganancia de tiempo cuando se precise disminuir

este rápidamente, y aquellos con un menor incremento de coste cuando se requiera minimizar dicho parámetro.

Por lo tanto, cada vez que se vaya a asignar un grupo a la partición hardware, sus nodos se ordenarán según el criterio comentado, y se irán moviendo sucesivamente, escogiendo su implementación y enlaces forzados de tal manera que siempre se tienda a la mayor optimización de la función objetivo.

6.3.3. Agrupamiento multi-etapa.

Hasta aquí se han presentado los dos procesos necesarios para realizar una etapa de agrupamiento previo sobre un algoritmo iterativo: la elección del criterio y el mapeado de nodos.

Gracias a esta etapa, se mejora el proceso de particionamiento, y se solventan las dificultades para las que antes no existía solución. Sin embargo, como ya se comentó en su momento, el proceso de agrupamiento puede convertirse en una metodología contraproducente.

En efecto, si el criterio elegido no ha sido eficazmente calculado, o lo que es lo mismo, se han agrupado comunicaciones indebidamente, el resultado obtenido puede llegar a ser peor que en el caso del particionamiento estándar.

De esta manera, puede ser que la mejor solución posible, para un *problema dado, se produzca situando dos nodos adyacentes en particiones* contrarias, y por lo tanto, activando la comunicación común. Si se ha considerado que esta comunicación tenía un efecto negativo sobre el sistema, y que debía ser agrupada, nunca será posible alcanzar la situación anterior, y por lo tanto, nunca se conseguirá llegar a la solución óptima.

Por lo tanto, un mal criterio de agrupamiento no sólo hace que no se pueda optar a una solución mejor que en el caso clásico, sino que puede conducir al particionador a una peor.

Desafortunadamente, nunca se puede garantizar que un criterio de agrupamiento sea eficaz, sin un estudio previo exhaustivo del problema y de sus condiciones iniciales, lo cual va en contra del principio de automatización del proceso.

El método de selección del criterio presentado en este capítulo ha ofrecido resultados muy favorables, lo cual no quiere decir que bajo ciertas circunstancias no sea insuficiente para llegar a un buen resultado, algo que no es posible predecir previamente.

Por lo tanto, y partiendo de la base de que el agrupamiento es algo positivo, sería conveniente idear algún método que solventara la elección de un mal criterio, eliminando sus efectos negativos.

El método que aquí presento está basado en los procesos que mediante sucesivas iteraciones refinan una solución dada, y se denomina *particionamiento con agrupamiento multi-etapa* [Maes96].

Si se parte de la base de que el método de selección del criterio de agrupamiento del que se dispone ofrece una calidad razonable, es de supone que el error al elegir dicho criterio es debido a la consideración como costosas de unas pocas comunicaciones, que de hecho no lo son.

Por lo tanto, el algoritmo de particionamiento no puede evolucionar de forma correcta, al haber colapsado estas comunicaciones dentro de grupos, evitando así la exploración de todas las soluciones en las que deben estar activas, o dicho de otra manera, en las que los nodos adyacentes deben ser asignados a particiones distintas.

Una vez que el particionador se haya decantado por una solución, es imposible determinar si esa es realmente la óptima (o una con una calidad aceptable), o sencillamente la mejor del espacio de búsqueda restringido por el agrupamiento previo.

Por lo tanto, y aunque esta solución pueda parecer satisfactoria, sería deseable ofrecer al particionador la posibilidad de explorar más allá del

punto en que lo ha hecho, con la intención de subsanar las posibles distorsiones creadas por una hipotética[7] mala elección del criterio.

De esta manera, y dado un criterio de agrupamiento, τ, *correspondiente a la primera comunicación del sistema que se colapsa*, surge la duda de si esta era realmente merecedora de esa acción, o si por el contrario era admisible su activación. Con esta última conjetura, se pasaría automáticamente al caso de que el criterio de agrupamiento se deba asociar a la *siguiente* comunicación, en orden de magnitud, dejando a esta *evolucionar libremente. Este hecho expande en parte el espacio de* búsqueda previamente restringido, y ofrece la posibilidad de encontrar nuevas soluciones mejores que la primera considerada. Así, se procede a una nueva iteración de la tarea de particionamiento.

Si en esta segunda iteración se encuentra una solución mejor, se *habrá avanzado respecto al caso de agrupamiento inicial, pero si no es así*, todavía se cuenta con el primer resultado, que se supone de una buena calidad.

En este segundo caso, en el que no se ha mejorado respecto de la primera iteración, no se puede afirmar libremente que ya no exista *posibilidad de mejorar. Esto es obvio, porque en todo método no exhaustivo* nunca se puede asegurar haber encontrado el óptimo.

Por lo tanto, en cualquier situación, se puede dar un paso más, y volver a relajar el criterio de agrupamiento, liberando la comunicación límite, y colapsando las superiores. De esta manera, se volverá a proceder igual. En

[7] Hipotética, porque *a priori* es imposible decidir si el criterio es adecuado o no, lo cual ocurre en todos los casos. Podría pensarse que ante tal incertidumbre, lo más adecuado es ignorar el proceso de agrupamiento y pasar directamente al particionamiento clásico. Es conveniente no caer en esta suposición, porque en dicho caso las posibilidades de no haber encontrado *la solución óptima sí son considerables.*

cada caso, se puede obtener una mejoría respecto del estado anterior o no, pero siempre es posible intentar realizar una iteración más.

Evidentemente, el límite será aquel punto en el que ya no existan más comunicaciones agrupadas, y por lo tanto, se hayan deshecho todos los grupos iniciales, quedando los nodos de nuevo visibles para el particionador. Este caso correspondería al particionamiento clásico.

Si se realiza una nueva iteración en estas condiciones, la solución final obtenida a lo largo de todo el proceso sí se puede garantizar que es la mejor respecto a la asociada al caso estándar.

Por una parte, no es peor que la hallada mediante esta situación, ya que este es un caso, concretamente el último, del agrupamiento multi-etapa, y por lo tanto ha sido explorado. Por otra parte, es posible que sea mejor, ya que se ha examinado mucho más espacio de diseño, y el proceso de agrupamiento puede haber ayudado a solventar los efectos debidos a comunicaciones elevadas.

Así, se ve que esta técnica conjuga las virtudes de ambos métodos, ofreciendo un mecanismo de resolución híbrido, que suele ser el mejor en la mayoría de los casos. Aunque pueda parecer que al final se ha acabado realizando un particionamiento estándar, no es eficiente optar por este desde un primer momento ya que se estaría obviando una gran parte del espacio de soluciones, donde puede encontrarse la óptima.

Un punto que se puede argumentar en contra de este sistema, es su aparente elevado tiempo de ejecución. Si se comentó que una simple ejecución del algoritmo de particionamiento puede acarrear un excesivo consumo de tiempo, tanto más lo hará una técnica que necesita varias iteraciones de este proceso para llevar a cabo sus operaciones.

Esta consideración es cierta en parte. La verdad es que en las primeras etapas del proceso, donde los nodos están fuertemente agrupados, y por tanto el número de entidades que se manipulan es menor,

el tiempo que se necesita para completar una iteración es pequeño comparado con el caso de no agrupamiento.

Así, a medida que el proceso avanza, el tiempo requerido es más elevado, pero también se ha explorado un mayor número de soluciones, y la considerada definitiva hasta ese momento debería estar próxima a la óptima, gozando de un alto nivel del calidad. Esto se debe, no olvidemos, a que se supone que el criterio de agrupamiento es razonable, como el presentado en este capítulo, y por lo tanto, una única iteración con este criterio debería ser más que suficiente. Los sucesivos pasos van encaminados a refinar aún más la solución, en el caso de que esto sea posible.

Este hecho sirve para comentar que aunque lo que se ha explicado es el caso límite de agrupamiento multi-etapa, no es necesario llegar siempre hasta ese punto. Dicho de otra manera, si todo se ha desarrollado de una forma conveniente, las últimas etapas no deberían ofrecer mejoras significativas respecto a las primeras iteraciones.

Por lo tanto, una buena alternativa sería realizar el proceso de agrupamiento, y posteriormente liberar un conjunto reducido de comunicaciones para solventar posibles deficiencias en la elección del criterio, con una inversión suplementaria en el tiempo de convergencia del algoritmo.

Faltaría por determinar cuántas iteraciones más allá de la inicial sería conveniente realizar, con el fin de que la relación entre el tiempo extra invertido y la calidad mejorada fuese óptima. Esto, obviamente, es difícil de predecir teóricamente. En el Capítulo 7 aparecen los resultados experimentales obtenidos y a partir de ellos se podrán extraer ciertas conclusiones a este respecto.

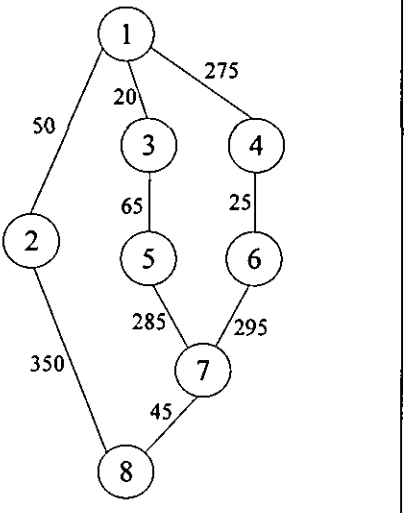


Figura 6.6

Modelado del ejemplo

6.4 Ejemplo de aplicación del proceso de agrupamiento a un sistema de Codiseño.

En este apartado se pretende dar, a partir de un ejemplo sencillo, una visión conjunta de los resultados obtenidos mediante la técnica de particionamiento estándar, frente al realizado con un proceso de agrupamiento previo. Para ello, se utilizará la misma estructura interna del grafo ofrecido en el Capítulo 4, al hablar de las características de la estimación mediante parámetros macroscópicos. Sin embargo, en este caso, los valores de las comunicaciones se han aumentado más, para estudiar su efecto global sobre el proceso. La estructura definitiva del ejemplo se ofrece en la Figura 6.6.

Por lo tanto, el esquema de trabajo sobre este ejemplo consiste en calcular las mejores soluciones obtenidas mediante el particionamiento estándar, para un rango amplio de valores asociados a la restricción

temporal, y compararlos, uno a uno, con el caso de realizar un agrupamiento previo.

Obviamente, en esta última situación, es preciso realizar el cálculo del criterio de agrupamiento para cada uno de los puntos, porque como se recordará, este criterio era dependiente del valor de la restricción temporal que se haya impuesto.

Es importante el hecho de que se considere todo el rango completo de restricciones, para que así este parámetro no polarice en exceso el estudio de los resultados.

Está claro que este rango de restricciones está limitado por los casos de tiempo máximo y mínimo que el circuito puede aceptar. El tiempo máximo estaría relacionado con el caso de ejecución de todos los nodos en software. El tiempo mínimo estaría asociado a la asignación de todos los nodos en hardware, con su implementación más rápida. Por lo tanto, será este rango de valores el que se estudie con el ejemplo.

Una vez determinado el propósito del experimento, el sistema sobre el que se va a realizar, y los dos métodos que se quieren comparar, es necesario comentar otro factor sin el cual no se puede llevar a cabo nada de lo anterior: la función objetivo.

6.4.1. Determinación de la función objetivo.

Esta función constituye una característica esencial dentro del particionamiento, no sólo porque interviene en todos los algoritmos de optimización existentes, sino por su relevancia conceptual.

En efecto, es esta expresión la que añade al proceso el conocimiento necesario para que mediante su minimización, siempre se tienda a la consecución de los objetivos marcados. Evidentemente, estos objetivos no serán siempre los mismos, y es preciso aclarar en cada momento cuáles son.

Así, en un determinado proceso, se puede querer optimizar prioritariamente el tiempo, desestimando la posibilidad de minimizar el coste. Sin embargo, pueden existir situaciones en las que se prime totalmente lo contrario. Además, es posible la intervención de otro tipo de parámetros, aparte de los de tiempo y coste que se están manejando aquí, como pueden ser los de potencia consumida o testabilidad del sistema.

Por lo tanto, la primera característica fundamental de la función objetivo es la conjugación de todos los factores trascendentes en el proceso. No obstante, es probable que en la mayoría de los casos no sea suficiente un único objetivo. Esto se debe a que todos los parámetros están relacionados entre sí, y la variación voluntaria de uno de ellos producirá una segura alteración involuntaria en otro.

Si, por ejemplo, se opta por un único criterio de minimización del tiempo, es posible que esto se consiga sin mucho esfuerzo, pero es más que probable la obtención de un resultado con un coste excesivamente alto. Por lo tanto, es siempre preciso delimitar cuál es el objetivo prioritario del proceso, pero además hay que explicitar el objetivo u objetivos secundarios, para mantener el resto de los parámetros dentro de unos márgenes razonables.

No sólo es preciso indicar todos estos objetivos, sino que además es necesario elegirlos de tal manera que exista el orden de relevancia deseado entre ellos. Por regla general, se dice que una solución es *válida*, si ha cumplido el objetivo prioritario impuesto. Sin esta condición, dicha solución ha de ser automáticamente descartada. Por otro lado, dadas varias soluciones válidas, se dice que una de ellas es *mejor* que el resto cuando ha cumplido en un mayor grado el objetivo secundario, lo que habitualmente está relacionado con una mayor optimización de un cierto parámetro.

Una última característica que debe cumplir la función objetivo es su simplicidad de cálculo. Evidentemente, esta función debe ser hallada cada

vez que el particionador tenga que decidir entre varias opciones, con el fin de comparar la calidad relativa de una solución respecto a otra. Por lo tanto, y debido a su gran uso, es preciso que no consuma demasiado tiempo a fin de que no repercuta sobre el proceso global de Codiseño en exceso.

Con todas estas consideraciones, una función de coste apropiada sería la siguiente:

$$f = \begin{cases} K \cdot (t - t_r) + c & \text{si } t > t_r \\ c & \text{si } t \leq t_r \end{cases} \qquad \textbf{[6.19]}$$

Como se observa, esta función consta de dos términos bien diferenciados. El primero evalúa la violación de la restricción temporal, $(t - t_r)$, y es cero en el caso de que el sistema haya cumplido dicha restricción ($t \leq t_r$). El segundo valora el coste global del sistema, c.

Un estudio de la función anterior deja claro que los dos objetivos impuestos al sistema están perfectamente definidos: el prioritario, de cumplimiento de la restricción temporal impuesta, y el secundario, de optimización del coste. Esto es así debido a la presencia de la constante K, la cual debe cumplir la propiedad de tener un valor absoluto lo suficientemente grande como para que el primer término de la expresión (la diferencia de tiempos) pese siempre más que el segundo (el coste).

De esta manera, cuando no se cumpla la restricción, nos encontraremos en la primera definición de la función, y al pesar el primer término más, será la violación temporal la que siempre se intente disminuir lo máximo posible. En el caso de dos soluciones con igual reducción de esta violación, decidiría el segundo término, el coste, y se optaría por aquella en la que dicho parámetro fuese menor.

Por el contrario, en el caso de que sí se cumpla la restricción temporal (segunda definición), el único factor influyente es el coste, el cual se trata de

minimizar. En este punto reside la idea de que una vez se haya obtenido una solución válida, la reducción subsiguiente del tiempo no es interesante, sino que lo que se prima es el abaratamiento del coste.

En cuanto a la simplicidad de la función, es evidente que se trata de una expresión fácil de calcular, ya que las únicas operaciones que intervienen son aditivas, aparte del producto por una constante.

Así, se comprueba que la función objetivo propuesta cumple las dos condiciones mencionadas en esta sección, lo que la hace idónea para guiar los procesos de particionamiento en Codiseño.

Por supuesto, cualquier otra función que igualmente respete estos compromisos sería válida, pero siempre con la certeza de que incorpora correctamente el conocimiento suficiente como para guiar al proceso hacia la mejor solución esperada.

6.4.2. Resultado del proceso de particionamiento.

Una vez presentado el grafo de trabajo y la función objetivo que se va a utilizar, se está en disposición de realizar el experimento propuesto en esta sección. El estado inicial del proceso corresponde a la situación en la que todos los nodos se encuentran en software.

El rango de valores de la restricción temporal, que como se comentó es máximo, corresponde al intervalo [65,540]. El primer valor corresponde a la ejecución de todos los nodos en hardware, con su implementación más rápida, y el segundo corresponde al caso en que todos los nodos están asignados a software (que corresponde a la situación inicial, y representa una solución trivial en la que directamente se cumplen las restricciones temporales).

Este intervalo se ha dividido en 20 puntos equidistantes, que corresponden a las distintas veces que el experimento se va a llevar a cabo. Sobre ellos, se calcula la mejor solución posible con cada una de las dos

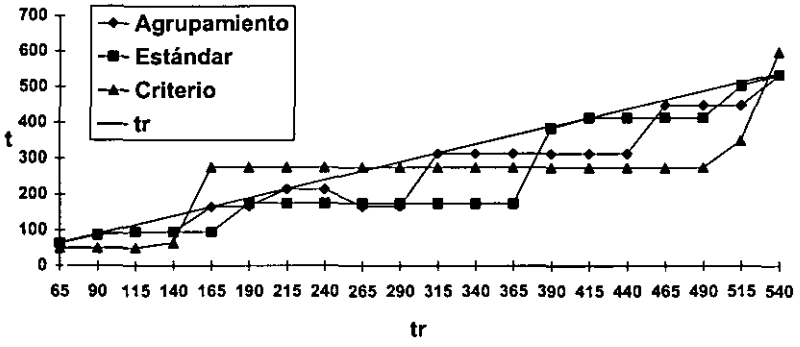


Figura 6.7
Resultados del tiempo

técnicas sujetas a examen: el particionamiento estándar y el particionamiento con agrupamiento previo. Para este último caso, se han escogido los valores de $c_1=c_2=1$ y $h=0.9$, para ajustar el proceso de selección de criterio.

En la Figura 6.7, se observa la representación de los tiempos de *ejecución asociados a las distintas soluciones halladas. Como se* comprueba, todas ellas son válidas para ambas técnicas, ya que siempre cumplen la restricción temporal impuesta. Esto se traduce en que las dos gráficas siempre discurren en una región inferior a la delimitada por la recta asociada a la restricción temporal.

Por otra parte, en la misma figura también se ha representado el criterio de agrupamiento, τ, hallado por esta técnica para cada uno de los casos de restricción temporal.

En la Figura 6.8 se pueden observar las dos gráficas de coste asociadas a ambas técnicas. Como se ha visto que todas las soluciones son válidas, y por lo tanto aceptables, es preciso estudiar el coste asociado a

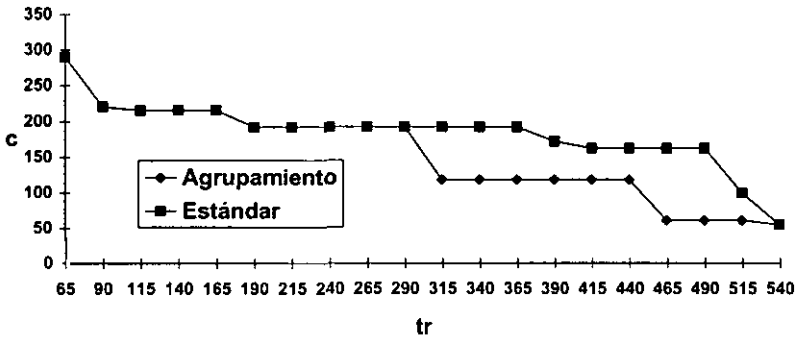


Figura 6.8

Resultados del coste

cada una de ellas para comparar, dos a dos, su calidad, y decidir cuál es la mejor.

La conclusión que se puede deducir de esta comparación es que la técnica con agrupamiento *siempre* encuentra una solución mejor, o en todo caso igual, que la técnica estándar, debido a que la gráfica de costes de la primera aparece siempre por debajo de la segunda. A partir de estos resultados es preciso realizar una serie de comentarios:

- La técnica con agrupamiento ofrece mejores soluciones cuando el valor de la restricción temporal, t_r, es alto, o dicho de otra manera, dicha restricción es relajada. Esto se debe a que en ese caso, el número de nodos que es preciso mover a hardware es relativamente bajo, ya que esta restricción no es excesivamente difícil de cumplir. Por lo tanto, el algoritmo de particionamiento se ve obligado a realizar pocos pasos, en los que las decisiones tomadas son críticas. En otras palabras, el algoritmo tiene menos oportunidades de compensar malas opciones tomadas en un primer momento, llegando rápidamente a un mínimo local. Por el contrario, si la restricción es fuerte y el algoritmo debe mover un alto número de nodos, existen más posibilidades de que reaccione ante

decisiones erróneas previas, con la consiguiente mayor facilidad de escape de mínimos locales.

- El criterio de agrupamiento τ se convierte en más estricto a medida que la restricción temporal se hace más fuerte. Para valores intermedios de esta restricción, τ permanece estable. Esto es debido a que existe un amplio vacío en los valores intermedios de las comunicaciones. En efecto, no aparece ninguna comunicación entre los valores 65 y 285, por lo que sucesivas variaciones de τ no tienen ningún efecto en el agrupamiento.
- Aunque no se muestra en las gráficas, el tiempo consumido por el algoritmo para producir los resultados es bastante menor en el caso de la técnica de agrupamiento que en el particionamiento estándar, casi del orden de la mitad. Esto se debe a que al agrupar, el número de entidades que el particionador debe manipular se ve reducido, con la consiguiente disminución del tiempo de convergencia.

En este ejemplo, y debido a que con una iteración del proceso siempre se han obtenido mejores soluciones en el caso del agrupamiento que en el caso estándar, no se ha realizado una extensión con el método multi-etapa. En el siguiente capítulo se podrán estudiar los experimentos detallados relacionados con este aspecto, incluido el comportamiento de esta técnica iterativa.

Hasta aquí se ha ofrecido el trabajo sobre particionamiento asociado a los métodos de estimación, incluyendo una técnica heurística automática de selección del criterio de agrupamiento, así como una expansión del concepto clásico de movimiento. Con ello, no sólo he querido aportar nuevas ideas al ya muy explotado campo del particionamiento, razón esta fundamental en el desarrollo de este capítulo, sino también dar una visión de la alta conectividad e interdependencia de todas las tareas que conforman el proceso de Codiseño.

Aplicación experimental de las técnicas Macroscópicas.

7

En los capítulos previos, he explicado con detalle la nueva visión macroscópica, así como las técnicas relacionadas en los campos de estimación y particionamiento. Aunque se ha tratado de demostrar que toda la teoría presentada es lógica y consistente, como así lo demuestra la formulación matemática asociada, puede y debe quedar alguna duda de su aplicabilidad, o en otras palabras, en cómo se comportaría al ser utilizada en casos experimentales. El hecho de que una aproximación sea racional, y que no contravenga los principios del ámbito de trabajo donde se desarrolle, no quiere decir que sea la mejor, ni siquiera que pueda llevarse a cabo de una forma eficiente.

De esta manera, ideas lógicas pueden existir muchas *a priori*, y la elección de una de ellas se corresponderá con criterios de simplicidad, o de proximidad a teorías ya conocidas, elementos estos totalmente válidos. Pero si esos criterios no ofrecen como resultado una argumentación válida en un entorno experimental, todas las suposiciones previas realizadas quedan sin efecto, siendo necesaria la proposición de modelos más refinados que sean compatibles con los problemas reales.

En consecuencia, es preciso la validación práctica de las técnicas macroscópicas elaboradas, a fin de contrastar de una manera definitiva su aplicabilidad. Así, y a lo largo de este capítulo, presentaré todo el trabajo

experimental realizado, tanto sobre las técnicas de estimación como sobre las llevadas a cabo en el particionamiento.

7.1 Presentación del entorno experimental.

Una de las características básicas de la labor experimental es que ha de tener una entidad suficiente como para que no queden dudas acerca de la fiabilidad de los resultados. Es decir, no es admisible una carga experimental de un tamaño reducido, ya que aunque los valores obtenidos sean aceptables, no es posible realizar una extrapolación hacia un conjunto general de problemas reales.

De esta manera, no es difícil encontrar trabajos que se sustentan en la aplicación de la teoría sobre un conjunto muy limitado de ejemplos supuestamente significativos. Es evidente que el uso de estos ejemplos es un hecho positivo que puede aportar claridad sobre el proceso general, debido a su nexo con el campo real de aplicación, pero la no extensión de los métodos hacia pruebas más elaboradas y numerosas, puede reducir las conclusiones extraídas al nivel de casualidad.

En efecto, muchas veces se presentan resultados experimentales adornados con el calificativo de *reales*[1]. Sin embargo, puede darse el caso de que las técnicas funcionen particularmente bien para el número limitado de ejemplos estudiados, y no tan bien para el resto.

Esta última afirmación no va encaminada a sugerir la posible polarización de los datos de entrada a los experimentos habituales, o en otras palabras, la elección subjetiva y sesgada de los ejemplos prácticos. Sencillamente me refiero al hecho de que cada problema es distinto, y existe

[1] Concepto este de *realidad* ampliamente subjetivo, aunque puedan parecer términos contrapuestos. No es difícil encontrar casos de ejemplos con un contenido vacío, cuya única vinculación con el mundo real es un *barniz* superficial.

la posibilidad de que los estudiados correspondan a un conjunto aparentemente más significativo, o más interesante desde algún punto de vista subjetivo. Sin embargo, pueden existir conjuntos de problemas totalmente diferentes, e igualmente necesarios de tratar, que se comporten de una manera radicalmente distinta a los elegidos.

Toda esta discusión va encaminada a proponer un banco de pruebas con un número elevado de experimentos, escogidos de tal manera que no quepa duda acerca de la objetividad en su elección, pero que además cumplan los requisitos necesarios como para ser considerados dentro del campo de investigación presente. Ambas condiciones darán la posibilidad de extrapolar, y por tanto de generalizar, los resultados que se hayan obtenido.

7.1.1. Características de la elección de experimentos.

En este punto, es posible optar por diferentes políticas de generación de pruebas experimentales para cada una de las distintas propuestas teóricas presentadas. Según lo comentado en los párrafos anteriores, las características básicas que deben poseer estas pruebas han de estar orientadas hacia una extrapolación global de los resultados, que permita una generalización de las conclusiones obtenidas.

Básicamente, es posible decantarse por dos aproximaciones bien distintas: por un lado, la extracción de problemas de entornos prácticos; y por otra parte, la consideración de pruebas generadas artificialmente, mediante algún método racional de elaboración de problemas.

Ambas técnicas poseen una serie de ventajas e inconvenientes. Sin embargo, se ha optado por la segunda, debido a la escasez de ejemplos detallados en la literatura, y a la alta complejidad que supondría el adaptarlos de aplicaciones reales. Esto redundaría en un reducido tamaño

del conjunto de pruebas experimentales, y por lo tanto en una escasa repercusión de los resultados obtenidos.

En consecuencia, creo que la utilización de una serie de pruebas generadas artificialmente es más adecuada y ventajosa para estudiar las características de las aproximaciones teóricas presentadas en este caso. Es obvio que, a raíz de haber adoptado esta política, existen distintas orientaciones para poder llevar a cabo la composición del banco de pruebas.

De entre ellas, la más objetiva, y también la más presente en la literatura actual, es la generación aleatoria de problemas [Kala95]. Quizás el hecho de introducir el concepto de aleatoriedad pueda inducir a la idea de falta de rigurosidad o de control sobre el proceso. Sin embargo, y si la tarea se realiza de una forma conveniente, no sólo se evitan los problemas anteriores, sino que además se obtienen una serie de ventajas fundamentales para poder explorar un amplio rango de problemas relacionados con el Codiseño.

Por supuesto, la generación de diseños de prueba ha de realizarse de tal forma que los casos obtenidos posean las características esenciales de los relacionados con el campo tratado. De esta manera, el proceso aleatorio no lo sería en su totalidad, sino que se dispondría de un mecanismo que forzara a los problemas creados a parecerse a aquellos que se pueden encontrar en los casos prácticos reales.

De esta manera, y asumiendo las múltiples posibilidades de abordar los procesos aleatorios, se puede llegar a la conclusión de que la aplicación de estos métodos a la verificación de la calidad de los sistemas de estimación y particionamiento es una decisión plenamente justificada, teniendo en cuenta una serie de ventajas intrínsecas:

- La aleatoriedad implica una objetividad de la que muchos diseños generados manualmente carecen. Hay veces en las que, si se quiere crear un objeto de prueba se tiende a la búsqueda de aquellos cuyas

características sean más apropiadas para ello. Si bien el uso de pruebas ventajosas no es algo criticable, sí es cierto que se puede polarizar excesivamente el conjunto de diseños hacia áreas muy restringidas, perdiendo generalidad en este proceso.

- Es posible generar un número mucho mayor de pruebas. Al ser el proceso totalmente automático, la producción de diseños se realiza en un tiempo mucho menor que el que llevaría una generación manual, producto del análisis de sistemas reales. El incremento del número de pruebas incide favorablemente en la calidad de la evaluación del sistema. Con unos resultados satisfactorios sobre tres o cuatro diseños no se puede deducir que el proceso de estimación sea globalmente correcto, algo que sí se puede concluir tras realizar varias decenas de pruebas.
- Se permite una versatilidad mayor en el proceso de prueba. Al ser la generación aleatoria automática totalmente parametrizable, es posible realizar prácticamente cualquier estudio sobre el diseño de prueba. Así, es realmente sencillo hacer análisis comparativos al variar el número de nodos del diseño, el número de etapas de cada nodo, la cantidad y tipo de unidades funcionales, la conectividad del grafo, etc.

El hecho de que un sistema sea real no implica que sus resultados sean más representativos que los de otro generado aleatoriamente. Sin embargo, el proceso de generación ha de cumplir una condición básica para que sus resultados puedan ser considerados válidos: tiene que representar de una forma fiel los sistemas típicos de Codiseño.

Con esto se quiere reseñar que si bien la *semántica* de los grafos generados no es trascendente para comprobar la calidad del estimador, su *estructura* o forma sí lo es, ya que este es un factor clave en el proceso de estudio.

De esta manera, diseños compuestos por un muy elevado número de módulos, o con parámetros hardware y software no equilibrados (por

ejemplo, tareas generadas que sean más rápidas en software que en hardware) pueden funcionar bien con el estimador, pero los resultados no serían extrapolables a casos reales, que tienen características bien distintas a las mencionadas anteriormente.

En resumen, el proceso de generación debería ofrecer diseños *isomorfos*, si bien no iguales, a los reales. Dicho de otra manera, aleatoriedad no quiere decir arbitrariedad. Esto se refleja en el entorno de generación de diseños de prueba que se ha desarrollado para servir de sustento a este proceso experimental.

Las posibilidades y funciones de este entorno se presentan con detalle en el Apéndice A. Quizás la propiedad más destacable introducida es la total libertad que posee el diseñador para generar problemas con un tipo de características fijadas, y así permitir la exploración de una cierta parte del conjunto de diseños existentes. De esta manera, es posible fijar fácilmente el número de nodos que el diseño ha de tener, el número medio de aristas, o el rango de valores de las comunicaciones generadas.

No obstante, y debido a la siempre necesaria existencia de un nexo entre las teorías desarrolladas y el mundo real, se ofrecerá un estudio concreto de las metodologías presentadas sobre un caso práctico basado en el estándar de codificación de vídeo H.261, con el fin de probar la adecuación de lo propuesto a casos más complejos (Véase Apéndice C).

7.1.2. Proceso experimental sobre las técnicas de estimación.

El primer aspecto experimental que se ha considerado es la obtención de la calidad de las distintas técnicas de estimación presentadas: la aproximación *plana*, la *escalar* y la *vectorial*. Para ello, se han realizado distintos estudios de comparación de los resultados hallados mediante estas técnicas, respecto al valor real medido en cada uno de los ejemplos.

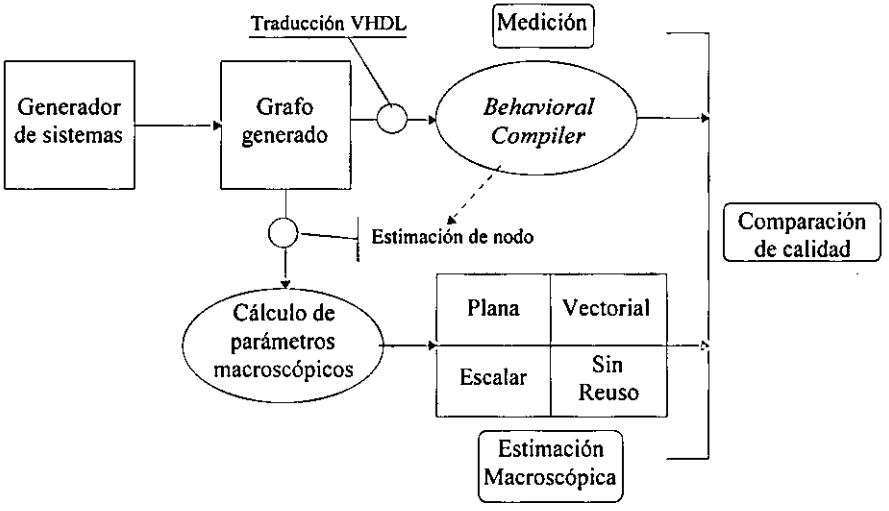


Figura 7.1

Esquema del proceso experimental sobre las estimaciones

Además de esto, se ha procedido al examen de la técnica trivial de estimación de costes aditivos. En esta técnica, no se considera ningún tipo de reuso hardware entre las distintas funcionalidades asociadas a esta partición. El hecho de tener en cuenta esta metodología reside en su todavía importante presencia dentro de múltiples aproximaciones en la literatura, a pesar de su excesiva simplicidad y falta de realismo. De esta manera, se obtiene un límite extremo de los valores de las estimaciones, que sirve como punto de referencia a la hora de estudiar cuánto se ha mejorado con la introducción de las nuevas técnicas presentadas.

Para el proceso de obtención del valor real de cada uno de los ejemplos, se ha utilizado la herramienta comercial de *Synopsys* denominada *Behavioral Compiler* (Véase Apéndice B). Mediante su uso, se han podido hallar los valores de tiempo y coste asociados a cada diseño, que

conformarán el conjunto de medidas reales respecto a las cuales poder calcular el error producido por el resto de las técnicas.

A continuación, aparecen los pasos seguidos en el proceso experimental de verificación de estas metodologías de estimación (Figura 7.1):

a) *Generación del grafo del problema.* El primer paso consiste en generar el diseño de prueba sobre el que se va a trabajar. Para ello, es preciso que previamente se hayan introducido las características deseadas para este diseño en el entorno de generación, referente a cada uno de los parámetros relacionados (Apéndice A). Una vez producido este hecho, se obtiene un grafo de tareas interconectadas, con la forma de la estructura interna presentada en §4.3.

Estas tareas se corresponden con los nodos asociados al grafo, que contienen a su vez el conjunto de operaciones internas. La generación de estas operaciones se realiza en función de la biblioteca de módulos introducida por el usuario, así como de la frecuencia seleccionada para cada una de ellas. De esta manera, cada uno de los nodos responde a las expectativas creadas en cuanto a su composición.

Por otra parte, como se ha venido afirmando a lo largo de este trabajo, es posible que cada una de las tareas pueda adoptar un número determinado de implementaciones, según se quiera que su actividad sea más o menos rápida. De esta manera, dependiendo del número de estas implementaciones que se haya seleccionado para cada tarea, se producirán tantos nodos asociados, todos ellos con las mismas operaciones (para que obviamente ejecuten la misma funcionalidad), pero con una distribución interna distinta.

Así, se tendrá que en la implementación más rápida existirán unos niveles de paralelismo elevados entre las operaciones, que se irán

eliminando a medida que las siguientes implementaciones sean más baratas, al introducir distintas etapas de control en los lugares apropiados.

De esta manera, al final de esta etapa, se tendrá un grafo interconectado, con las tareas expresadas en forma de nodo, cada uno de los cuales con la posibilidad de contar con más de una implementación.

b) *Traducción a VHDL.* Si se quiere, como ya ha sido comentado anteriormente, que los diseños generados sean procesados por el *Behavioral Compiler*, es preciso traducirlos a un lenguaje comprensible por esta herramienta. Se ha optado por VHDL, debido a su regularidad al especificar sistemas de alto nivel. De esta manera, se ha desarrollado un traductor de las tareas expresadas en la estructura interna como listas de operaciones, a VHDL de comportamiento.

La descripción generada está realizada a nivel RT, y consta de unos puertos de entrada, otros de salida, y un conjunto de registros para almacenar los valores intermedios de la tarea a través de las distintas etapas de control. Todos estos elementos se unen a las unidades funcionales mediante las correspondientes señales internas generadas.

Este proceso de traducción se realiza para cada uno de los nodos del grafo, con cada una de las implementaciones posibles, obteniendo el código VHDL asociado a todos estos casos. Es importante el hecho de que esta traducción se haya automatizado, debido al elevado tiempo que consumiría el tratarla de una manera manual.

c) *Realización de las estimaciones de nodo.* Ahora, para cada una de las descripciones halladas en el paso anterior, es preciso calcular los parámetros de tiempo y de coste asociados. Este paso, previo al comienzo de la metodología de Codiseño propiamente dicha, es el que sirve para caracterizar de una manera macroscópica el conjunto de estimaciones individuales *P*, presentado en §4.1.2. De esta manera, se obtienen los

distintos pares de tiempo y coste asociados a cada implementación y cada nodo existente en el diseño.

Esta caracterización resulta de vital importancia, ya que las futuras estimaciones sobre los parámetros globales del sistema se realizarán basándose en estas mediciones.

Otro punto importante es que el *Behavioral Compiler* podría haber alterado la planificación de las operaciones contenidas en los nodos, que previamente fueron propuestos. Así, la herramienta puede haber encontrado una planificación mejor que la considerada inicialmente, y que resulte en un menor coste para el tiempo de ejecución prefijado.

Es obvio que entonces es necesario adoptar esta planificación, por lo que los cambios realizados en este sentido han de ser transmitidos a la estructura interna que está siendo considerada por el sistema. De esta manera, se produce una realimentación de datos, y se pasa a adoptar la nueva distribución de operaciones. Así, el *Behavioral Compiler* no sólo trabaja en la realización de medidas, sino que tiene una importante función optimizadora.

d) *Cálculo de la semejanza*. Una vez determinados los parámetros asociados al número de implementaciones hardware, al tiempo y al coste, falta todavía realizar el cálculo del último parámetro intrínseco a los nodos, la semejanza (§4.1.3). Para ello, y según su definición, se procede a hallar el hardware compartible entre cada uno de los posibles pares de nodos, con cada una de las implementaciones existentes. Al disponer de todas las tablas de distribución de operaciones, esta tarea se realiza automáticamente sin ningún tipo de dificultad conceptual.

Al finalizar esta etapa, se dispondrá de la estructura interna del sistema generado en forma de grafo, con los distintos nodos caracterizados mediante el conjunto de parámetros intrínsecos, y con una distribución de operaciones optimizada mediante el *Behavioral Compiler*.

e) *Selección del punto de trabajo.* El siguiente paso está relacionado con la determinación del estado parcial de los nodos, para conformar una solución intermedia del sistema. Esto se corresponde al cálculo del parámetro extrínseco de estado, y equivale a la asignación a cada nodo de una partición determinada, una implementación y una posición relativa dentro de esta partición.

Esta información, como ya se vio en §6.2.2, vendría proporcionada por el particionador, que se encargaría de la exploración del espacio de soluciones, y por lo tanto correspondería a una situación intermedia del proceso de particionamiento.

Evidentemente, según el estado seleccionado, el sistema *global* tendrá asociados unos parámetros de tiempo y coste propios, que será la información que habrá que estimar mediante las distintas técnicas presentadas.

De esta manera, se realizarán los cálculos utilizando la aproximación plana, la escalar, la vectorial y la básica sin reuso, obteniendo previsiblemente distintos resultados para cada uno de los casos.

f) *Estudio de los resultados y repetición del proceso.* Una vez realizado el proceso de estimación de tiempo y coste según las distintas técnicas propuestas, es preciso comparar los resultados con las mediciones reales del *Behavioral Compiler*. Para ello, se introducirá la especificación completa del grafo, y se procederá al cálculo de los parámetros asociados, tomando estos como valores reales.

Sin embargo, las carencias propias de esta herramienta hacen que no sea posible la realización de esta operación directamente. En efecto, el método lógico con el que abordar esta tarea sería la traducción de cada uno de los nodos a una entidad distinta del diseño, y posteriormente proceder a su estimación. Sin embargo, el *Behavioral Compiler* no contempla la posibilidad de reuso hardware entre las distintas entidades, con lo que se

estaría descartando una de las opciones básicas de diseño introducidas en las técnicas de estimación. Por lo tanto, los resultados hallados no serían comparables, debido a las distintas consideraciones utilizadas en su obtención.

En consecuencia, se ha optado por la realización de un proceso de mezcla de todos los nodos a una única entidad VHDL, de una forma automática. Las descripciones generadas anteriormente para la realización de la estimación de nodos individuales son mezcladas dos a dos en un proceso iterativo, respetando los distintos valores de planificación para cada uno de los nodos.

Así, si en la estructura interna aparece que los tiempos de comienzo de ejecución de dos tareas en concreto se encuentran desfasados un cierto número de unidades de tiempo, esa diferencia se reflejará en el proceso de mezcla, de tal manera que la descripción final sea fiel imagen del problema considerado. Posteriormente, se procede a la medición de los valores de tiempo y coste asociados.

Aquí acabaría el proceso experimental de un estado particular asociado a un diseño concreto. Para tener una base de resultados fiable, es preciso repetir toda esta metodología, con la elección de sucesivos puntos de trabajo (paso e). Una vez que se haya procedido con un número significativo de estos puntos, hay que retomar el comienzo de la experimentación, y volver a generar un nuevo diseño (paso a), para el que a su vez se repetirá todo el proceso comentado.

Una vez concluido todo el ciclo de producción de resultados, se pasa a la etapa de estudio de las conclusiones. Es el momento de comparar los valores ofrecidos por las distintas técnicas de estimación con los valores reales medidos mediante el *Behavioral Compiler*. Los estudios realizados giran alrededor de la precisión, fidelidad y tiempo de ejecución asociados a cada una de las técnicas, que se ofrecerán posteriormente en §7.2.

Con esto finaliza el proceso experimental asociado a las metodologías de estimación propuestas.

7.1.3. Proceso experimental sobre las técnicas de particionamiento.

El segundo desarrollo experimental tiene como objeto de estudio los mecanismos de particionamiento presentados en §6, con la introducción de procesos de agrupamiento previos, y la posibilidad de extensión multi-etapa.

De esta manera, se pretende la comparación de estos resultados con el caso tradicional de particionamiento mediante algoritmo estándar, para así extraer las ventajas que de estas mejoras se deriven. Dichas ventajas han de ir en la dirección de minimizar el tiempo de ejecución destinado al proceso, a la vez que se incrementa el número de soluciones óptimas encontradas.

Para realizar estos experimentos se ha contado con el entorno de generación de sistemas presentado en la sección anterior, si bien la metodología seguida es distinta en ambos casos. Los pasos destinados a la obtención de esos procesos de prueba son los siguientes (Figura 7.2):

a) *Generación del grafo representativo del problema*. Para llevar a cabo esta operación, se utiliza el sistema generador de grafos aleatorios ya presentado. Así, y de la misma manera que ocurría en el proceso de prueba de las técnicas de estimación, se obtiene una estructura interna formada por un grafo de tareas interconectadas.

Estas tareas están contenidas dentro de los distintos nodos del grafo, y estarán caracterizadas mediante los parámetros macroscópicos adecuados. En consecuencia, la representación interna será perfectamente asumible por las técnicas de estimación, que podrán predecir los valores globales de tiempo y coste relacionados.

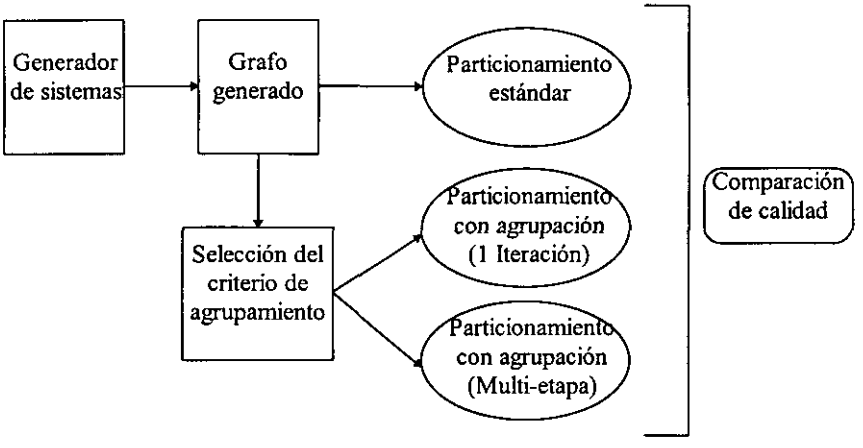


Figura 7.2

Esquema del proceso experimental sobre el particionamiento

b) *Elección de la restricción temporal.* Ahora, es preciso seleccionar un valor concreto de la restricción temporal t_r, que formará parte de la entrada al proceso de particionamiento. Obviamente, esta restricción ha de tener un valor razonable para que el problema de Codiseño tenga sentido, por lo que deberá estar comprendida entre los tiempos máximo y mínimo de ejecución del sistema. De no ser así, el problema tendrá una solución trivial, o no existirá tal solución, careciendo ambos casos de interés.

Por lo tanto, y para calcular estos límites, se halla el tiempo asociado al caso en que todos los nodos se ejecuten secuencialmente en la parte software (máximo), y el tiempo consumido cuando todos los nodos se encuentran en la partición hardware, con sus implementaciones más rápidas y el máximo grado de paralelismo posible (mínimo). De esta manera, y en función de ambos valores obtenidos, se elige un punto intermedio de la restricción temporal.

c) *Realización del proceso de agrupamiento y particionamiento posterior.* En este punto, se comienza con la obtención de resultados experimentales propiamente dichos. Primeramente, se calcula el criterio de agrupamiento de una forma automática, según se vio en §6.3.1.

De esta manera, y una vez hallado este límite de agrupamiento, se procede a agrupar el sistema obteniendo un nuevo grafo transformado, en el que los distintos nodos se han asociado para formar un número determinado de grupos.

A continuación, y sobre este nuevo grafo, se realiza el proceso de particionamiento clásico, con la consiguiente obtención de una solución asociada a la restricción impuesta. Obviamente, es de esperar que esta solución disponga de un tiempo de ejecución igual o menor que dicha restricción, con lo que se aceptará como válida.

En consecuencia, el valor del coste obtenido es lo que se considerará como futuro nivel relativo de calidad, ya que constituye la prueba de cómo y con qué exhaustividad se ha explorado el espacio de soluciones. Por otra parte, un dato relevante será el tiempo de cálculo de la solución por parte del entorno. Por supuesto, se pretende la obtención de buenos resultados, pero sin que el tiempo destinado a ello se dispare en exceso.

A este proceso de particionamiento con formación previa de grupos, se le denomina *agrupamiento con una iteración*, ya que no se exploran alternativas más allá del criterio obtenido en primera instancia.

d) *Realización del proceso de agrupamiento multi-etapa.* Como se comentó en su momento, la consideración de un criterio de formación de grupos no adecuado puede conducir a la obtención de peores resultados que en el caso de particionamiento estándar.

Para solventar esta deficiencia, se presentó el caso de agrupamiento multi-etapa, en el que partiendo de la solución obtenida por el primer criterio

hallado, se desagrupaban ciertos nodos, mediante una relajación de este criterio, repitiendo dicho proceso hasta un límite decidido por el usuario.

Evidentemente, este límite será finito, y corresponderá al caso en el que ya no existan más nodos por desagrupar, que estará asociado a la situación clásica en la que no se aplica ningún proceso de agrupamiento.

Hasta llegar a este umbral, existen situaciones intermedias en las que, a costa de dedicar un mayor tiempo, se explora el espacio de soluciones de una forma más exhaustiva. De esta manera, se han escogido una serie de puntos medios que permiten el estudio de la evolución de la calidad.

Así, y partiendo del caso presentado antes, correspondiente a una única iteración, se repite el proceso de particionamiento, relajando el criterio de formación de nodos, hasta el límite máximo posible, que se denominará *agrupamiento con número máximo de iteraciones.*

Este resultado servirá como punto de comparación con el obtenido anteriormente. Además de este caso, se han obtenido otros dos, en los que no se ha llegado al umbral máximo, sino que se ha decidido parar en situaciones intermedias, concretamente aquellas correspondientes al *agrupamiento con número de iteraciones igual a un medio del máximo*, y al *agrupamiento con número de iteraciones igual a tres cuartos del máximo.*

De esta manera, y como resumen, se obtienen cuatro valores para la solución hallada por el particionador al realizar cuatro procesos de agrupamiento multi-etapa, entre los que se incluyen los casos extremos de una iteración y máximo número de iteraciones.

e) *Realización del proceso de particionamiento clásico.* Con el fin de comparar las ventajas obtenidas por los casos anteriores, se calcula el resultado obtenido por el proceso de particionamiento estándar, sin realizar ningún agrupamiento previo. Además, también se calcula el tiempo consumido para la generación de la solución, a fin de proceder a un estudio posterior.

f) *Repetición del proceso y tratamiento de los resultados.* Todo lo expuesto hasta ahora corresponde al caso experimental de un único diseño, con una única restricción temporal. Ahora, es preciso variar esta restricción temporal dentro del rango calculado en etapas previas, de tal forma que se barra todo el espectro posible de sus valores, desde los más relajados a los más estrictos. De esta manera, se obtendrá un conjunto de resultados extenso, que pondrá de manifiesto las características del particionamiento sobre el sistema considerado.

Posteriormente, es preciso repetir la totalidad del proceso hasta este punto, pero generando un nuevo diseño aleatorio. Al final, se tendrá un conjunto de soluciones para una serie de diseños de prueba, cada uno de ellos con un amplio espectro de restricciones temporales asociadas.

Por último, será preciso realizar un estudio sobre los resultados obtenidos, a fin de extraer conclusiones extrapolables. Para ello, se procederá a comparar el número de soluciones mejores obtenidas por los distintos procesos de agrupamiento multi-etapa, frente al caso de particionamiento estándar. Asimismo, se realizará el estudio de soluciones iguales y peores bajo las mismas circunstancias. Esto dará una medida de la calidad de las distintas técnicas.

Además, se estudiará no sólo cuántas soluciones mejores se han obtenido, sino por cuánto margen de diferencia, lo que proporcionará el índice general de riesgo producido al adoptar un proceso de particionamiento frente a otro.

Finalmente, se llevará a cabo un estudio del tiempo consumido por cada una de las aproximaciones para obtener el resultado. Este tiempo será fundamental, ya que constituye el factor crítico dentro del proceso de Codiseño. Así, no será conveniente la utilización de una técnica muy ventajosa en cuanto a su calidad, si ello representa retrasar el proceso de una forma considerable.

7.2 Resultados experimentales obtenidos para el proceso de estimación.

7.2.1. Valores de contorno utilizados en el proceso.

La primera información dentro del apartado de presentación de los resultados experimentales debe ir referida a los *valores de contorno*. Esta expresión va asociada a todas aquellas características que delimitan parámetros variables dentro del proceso, como son las características de generación de los grafos, o las distintas bibliotecas de módulos utilizadas. Esta información es necesaria para entender el proceso experimental en su totalidad, así como para que los problemas tratados puedan ser reproducidos.

De esta manera, el número de experimentos efectuados se eleva hasta 204. Cada uno de estos experimentos comprende la labor de estimación de un diseño con un punto de trabajo concreto por parte de las distintas técnicas, y la correspondiente medición mediante el *Behavioral Compiler*.

Estos 204 experimentos se agrupan alrededor de 18 grafos de diseño distintos, de tal manera que se han realizado entre 8 y 18 pruebas (elecciones de puntos de trabajo) sobre cada uno de estos grafos. El hecho de considerar más o menos experimentos para cada uno de los distintos diseños viene dado por las características naturales de estos. Hay grafos con un espacio de diseño mayor, y por lo tanto con más altas posibilidades de estudio de distintos puntos, y hay otros a los que les ocurre totalmente lo contrario. De esta manera, y estimando unos umbrales máximo y mínimo para cada diseño, se ha obtenido aleatoriamente el número de experimentos realizados sobre cada grafo.

Los 18 grafos generados aleatoriamente se corresponden a tres series, constando cada serie de 6 grafos con un número total de nodos comprendido entre 7 y 12.

Cada uno de estos grafos, aparte del número de nodos ya comentado, dispone de una serie de parámetros introducidos al generador aleatorio, por los cuales adopta una forma y propiedades determinadas. Véase el Apéndice A para la definición de todas las características enunciadas a continuación.

Por un lado, un factor importante es el relacionado con la conectividad del grafo, o número medio de aristas existentes entre los nodos. Para esta experimentación se ha elegido un valor aleatorio de la conectividad entre 2 y 4. De esta manera, los grafos dispondrán de un número de aristas salientes comprendido entre estos valores, siempre que las circunstancias los permitan (por ejemplo, el nodo final siempre debe tener cero aristas salientes).

Los valores asociados a las comunicaciones se han elegido de forma que oscilen entre el 10% y el 100% del tiempo empleado por la tarea software adyacente, con un factor de desviación de entre un 0.0 y un 0.5.

Para la simulación de la parte software, que como se explicó en §2.2.3 viene caracterizada por un modelo simplificado, se han escogido unos parámetros de coste nulos. Para los parámetros de tiempo, se ha decidido que sean entre 10 y 20 veces mayores que la media de los parámetros asociados a las implementaciones hardware. El valor exacto entre estos dos límites se ha elegido de forma aleatoria. Sobre el tiempo hallado para la tarea software, se ha aplicado un mecanismo de corrección de entre el -10% y el +10%, también de forma aleatoria. Esto se realiza para introducir un pequeño factor de incertidumbre dentro del modelo, con el fin de que las estimaciones no sean excesivamente regulares.

Símbolo	U.F.	Tiempo (ns)	Coste (p.e.)
+	Sumador	17	85
-	Restador	12	88
*	Multiplicador	21	869
/	Divisor	76	528
#	Multiplicador-Acumulador	23	1194

Figura 7.3

Biblioteca de unidades utilizada

El número de implementaciones hardware ha sido generado aleatoriamente para cada nodo, con límites comprendidos entre 1 y 4. Por supuesto, como ya se comentó, los parámetros hardware asociados vienen medidos mediante el uso del *Behavioral Compiler*. El número de etapas de control para cada nodo ha sido generado entre 8 y 12. De igual forma, el número de unidades funcionales por etapa ha sido escogido entre 2 y 10.

La biblioteca utilizada es un subconjunto real de la biblioteca *class* contenida en el *Behavioral Compiler*. Se ha optado por un modelo simplificado para no complicar en exceso el proceso experimental, aunque todas las metodologías son igualmente aplicables a un modelo más extenso. Para este caso, se ha elegido una biblioteca compuesta por 5 unidades funcionales, que se corresponden con los tipos más frecuentes empleados en los problemas de Codiseño. Estas unidades funcionales son un sumador, un restador, un multiplicador, un divisor y un multiplicador-acumulador. Los valores de tiempo y coste proporcionados para cada unidad funcional por el *Behavioral Compiler* aparecen en la Figura 7.3. No se ha considerado la presencia de más de una unidad funcional que sea capaz de realizar la misma operación, por motivos de simplicidad, pero de nuevo con la

puntualización de que su inclusión se podría realizar sin ningún problema conceptual.

Las frecuencias de aparición de unidades funcionales empleadas en el generador de grafos han sido calculadas aleatoriamente para cada diseño.

Por último, el valor del ciclo de reloj se ha seleccionado como 80 ns. Esta decisión viene dada para que cualquier unidad funcional de la biblioteca tenga tiempo suficiente para ejecutarse dentro de los límites de uno de estos ciclos. Por supuesto, para casos reales, se necesitaría un estudio más exhaustivo de esta característica.

Con todos estos datos, la generación de grafos ha quedado totalmente caracterizada, así como el conjunto global de experimentos realizados. El último detalle está referido a la máquina, una Ultra-Sparc, donde se han llevado a cabo estos procesos, y sobre la que se han medido los distintos tiempos empleados en la consecución de los diferentes resultados.

7.2.2. Estudios prácticos realizados.

Según Gajski [GDWL92], las características básicas de un buen sistema de estimación deberían ser aquellas que condujeran a la obtención de resultados precisos, fiables, y que consuman poco tiempo en su producción. De esta manera se ha optado por estos criterios de estudio para calibrar las técnicas propuestas.

A continuación, se explicitan dichos criterios de la forma exacta en la que han sido tomados para su aplicación.

a) *Precisión.* Mide la proximidad de un valor estimado a su equivalente valor real. En otras palabras, indica el error relativo producido por la estimación. Cuanto más pequeño sea este error, se dirá que la técnica es más precisa. De esta manera se tiene que:

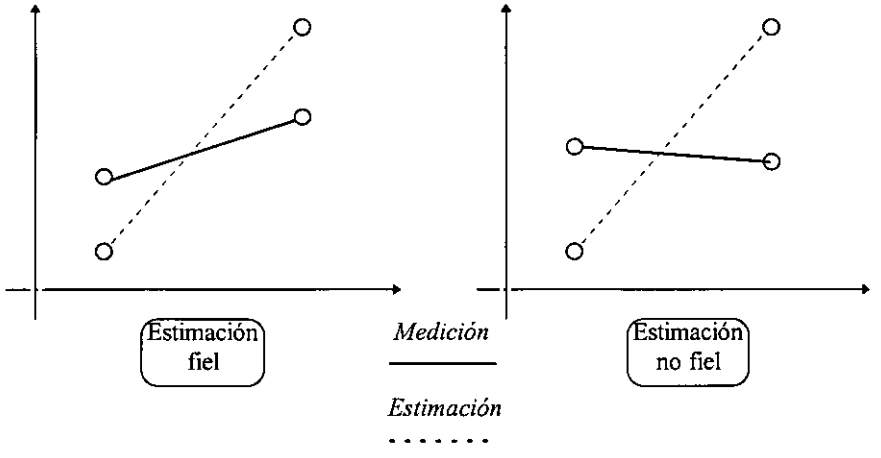


Figura 7.4

Fidelidad de la estimación sobre un par de puntos

$$e = \frac{v_e - v_r}{v_r} \qquad [7.1]$$

En esta expresión e indica el error relativo de precisión, v_e el valor estimado y v_r el valor real o verdadero asociado.

b) *Fidelidad*. Indica la comparación entre la calidad relativa de un par de puntos, según el valor estimado y según el valor real. Así, si dados dos puntos, la técnica de estimación predice que uno de ellos es mejor que el otro, y esta suposición coincide con el caso real de los valores medidos, entonces se dice que la técnica es fiel para ese par de puntos. (Véase Figura 7.4).

De esta manera, para comprobar la fidelidad global de la técnica, se calculan todos los posibles pares entre los puntos estimados, y se recuentan cuántos de estos pares han obtenido una estimación fiel y cuántos no. El porcentaje de casos fieles respecto del total posible, representará la

fidelidad global de la técnica de estimación dada en ese sistema. Este parámetro es de vital importancia dentro de los procesos de particionamiento, ya que en estos lo que se realiza es una comparación de la calidad relativa de dos soluciones. Así, lo importante no es conocer con exactitud el valor real asociado, sino sencillamente escoger la solución más conveniente en cada momento, respecto a otra propuesta.

c) *Tiempo de proceso.* Este tiempo corresponde al necesitado por la técnica de estimación para calcular los resultados sobre un punto dado. Es obvio que cuanto menor sea este tiempo, más ventajoso será el proceso en sí. De esta manera, puede haber técnicas que obtengan resultados positivos en cuanto a precisión y fidelidad, pero que para ello hayan necesitado consumir un elevado tiempo. En estos casos, las técnicas ya no son tan interesantes, debido a que pueden convertirse en inviables al ser integradas dentro de un proceso de particionamiento complejo.

Tras la explicación de los estudios realizados sobre los experimentos, es preciso pasar a detallar los resultados obtenidos en el proceso.

7.2.3. Relación de los resultados obtenidos.

El primer punto concerniente a los resultados experimentales relacionados con el proceso de estimación ha sido la obtención de los parámetros de tiempo y coste[2] asociados a cada uno de los 204 experimentos realizados, mediante las técnicas plana, escalar, vectorial y sin reuso, así como la medición con el *Behavioral Compiler.*

[2] A partir de este punto, todos los valores de tiempo están ofrecidos en ciclos de reloj (excepto los tiempos de proceso de las aproximaciones, que vienen dados en segundos), y los de coste en puertas equivalentes.

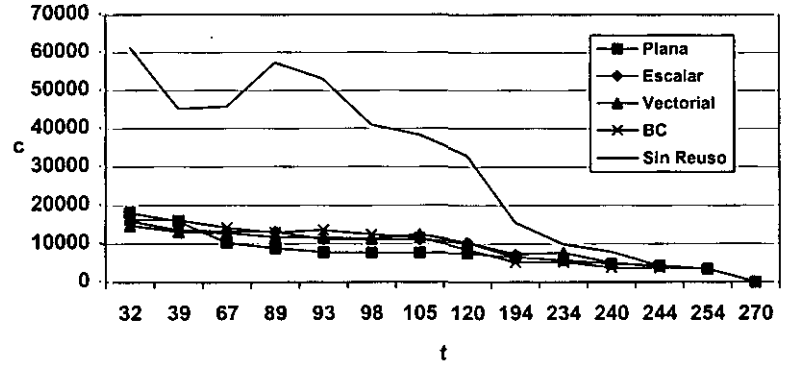


Figura 7.5

Grafo 6: Resultados comparativos de la técnica sin reuso

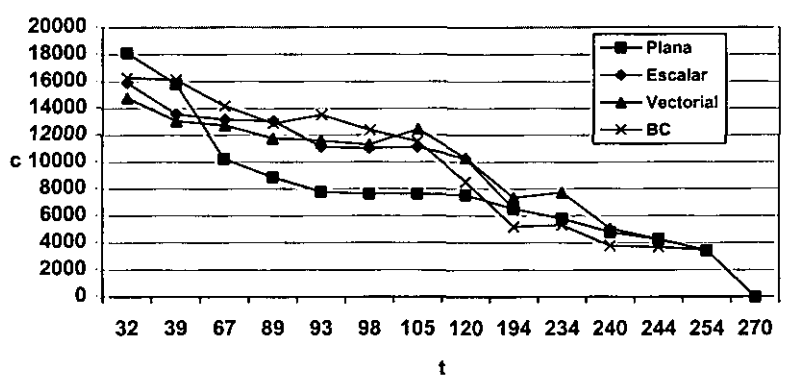


Figura 7.6

Grafo 6: Resultados comparativos con el *Behavioral Compiler*

De esta manera, y debido a que la técnica de estimación del tiempo asociado a un sistema es equivalente en todas las aproximaciones, las diferencias obtenidas versan alrededor del coste asociado, que es lo que determinará la calidad relativa del método

utilizado. Obviamente, detallar todos y cada uno de estos resultados es una tarea poco conveniente, debido a la facilidad de perder la visión general del proceso en una maraña de cifras. Por lo tanto, se ha tratado de simplificar la presentación de los valores obtenidos dentro de lo posible, ofreciendo gráficas equivalentes a fin de hacer más comprensible la información.

Los resultados se ofrecerán referidos a cada grafo, y vendrán representados por la media de los valores obtenidos en los puntos asociados.

De esta manera, y cuando se mencionen distintos resultados, como la precisión o el tiempo de ejecución de un grafo concreto, me estaré refiriendo a la media de los valores individuales obtenidos para las distintas soluciones posibles de ese diseño. Así, la información se mantiene más agrupada y manejable, pero es preciso tener siempre presente que estos valores provienen de la realización de varios experimentos.

Para comenzar, se ofrecen los resultados del coste obtenidos por cada una de las técnicas, al tratar distintos puntos de diseño con un tiempo de ejecución variado. Con el fin de no recargar la exposición con demasiados datos, he escogido tres grafos concretos de entre los dieciocho tratados, de tal manera que sus resultados se ofrecen en las Figuras 7.5 a 7.10.

Las Figuras 7.5 y 7.6 corresponden al grafo 6. La primera, representa todos los valores obtenidos por cada una de las técnicas. Como se observa, el orden de magnitud de la técnica sin reuso es considerablemente mayor que el resto, incluida la medición real con el *Behavioral Compiler* (BC). Este es un resultado lógico, ya que al no considerar la posibilidad de compartir hardware entre unidades, cada vez que se asigna un nodo más a hardware, el incremento de coste necesario para implementar sus funcionalidades es mucho mayor que si se hubiese reusado parcialmente.

Se ve, que esta diferencia se acentúa en gran manera cuando la mayoría del sistema se encuentra en hardware (parte izquierda de la

gráfica), y que tiende a desaparecer cuando casi todos los nodos están en software (parte derecha[3]). Este cambio brusco de orden de magnitud se corresponde a lo esperado, debido a que el efecto sobre el coste del sistema, si no se considera reuso, es muy alto cuando se tiene un número elevado de nodos hardware.

Dado que en la Figura 7.5 es difícil estudiar el resto de las técnicas, se repiten los resultados en la Figura 7.6, pero ahora sin mostrar la metodología sin reuso.

La conclusión que se puede extraer de dichos datos es que las técnicas presentadas ofrecen unos valores muy próximos a la medida real. Todas ellas oscilan alrededor de esta medida, sin llegar a alejarse demasiado, por lo que marcan de una manera correcta la forma de la evolución del coste. Esta evolución está asociada a la fidelidad del sistema, que por lo tanto se encuentra fuertemente acentuada en cualquiera de las tres aproximaciones.

Se comprueba, como es obvio, que a medida que el coste es menor, debido a una alta presencia de nodos asignados a software, la diferencia entre todas las técnicas disminuye, ya que la repercusión introducida por el reuso es menos influyente.

En la parte izquierda de la gráfica, donde es más difícil la predicción del valor correcto, la aproximación plana estima en general un conjunto de valores ligeramente por debajo de los correctos. Esto equivale a una sobrestimación del reuso, debido fundamentalmente a la no consideración de la estructura interna de los nodos.

[3] El punto de más a la derecha siempre corresponde a la situación en que todos los nodos están en software, y por lo tanto tiene un coste asociado de cero. Estos valores triviales no se han considerado en ningún caso para hacer ningún tipo de media, y no forman parte del conjunto experimental presentado.

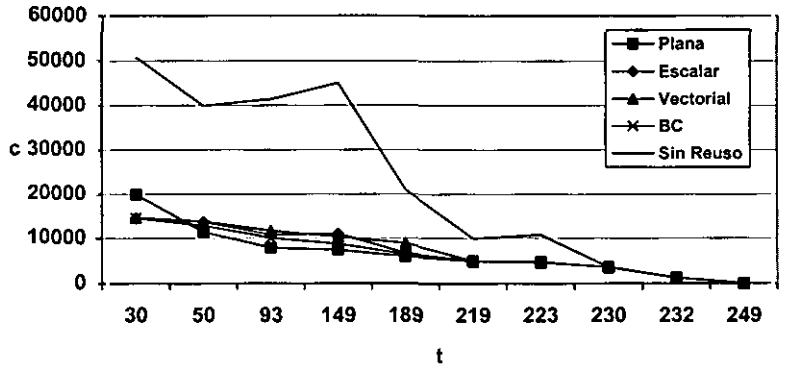


Figura 7.7

Grafo 11: Resultados comparativos de la técnica sin reuso

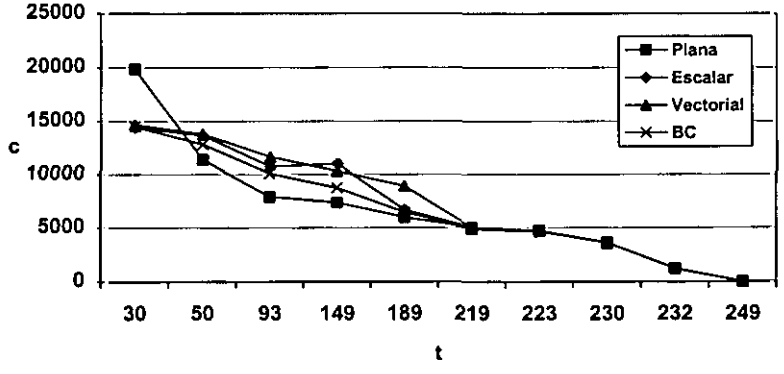
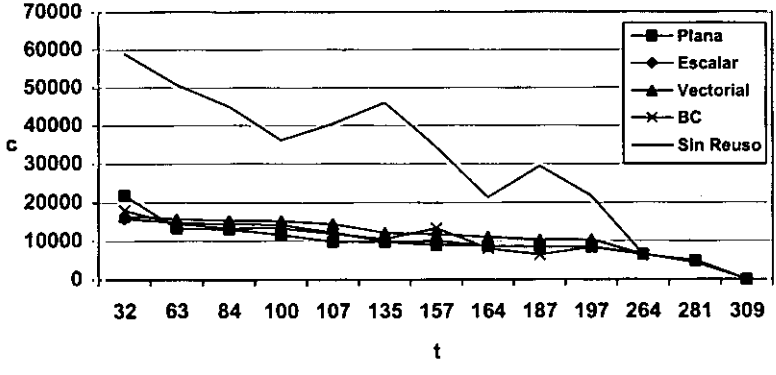


Figura 7.8

Grafo 11: Resultados comparativos con el *Behavioral Compiler*

Por otra parte, la aproximación escalar ofrece valores más parecidos a los reales que la vectorial, observándose en esta última un



.Figura 7.9

Grafo 15: Resultados comparativos de la técnica sin reuso

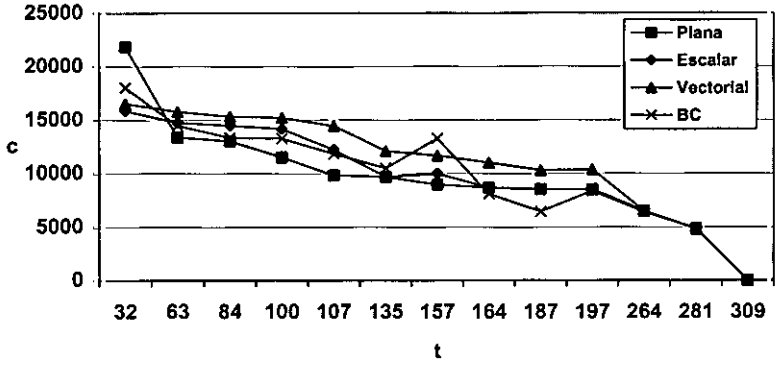


Figura 7.10

Grafo 15: Resultados comparativos con el *Behavioral Compiler*

cierto fenómeno oscilatorio, en el que se predicen alternativamente valores por exceso y por defecto.

La misma información, separando los casos en los que se muestran los resultados de la técnica sin reuso de aquellos donde no aparecen, se encuentra referida al grafo 11 (Figuras 7.7 y 7.8) y al grafo 15 (Figuras 7.9 y 7.10). Se ve que se mantienen las mismas tendencias marcadas en párrafos anteriores.

Lo mismo ocurre con el resto de grafos no explicitados, de tal forma que esta regularidad en los resultados implica una fiabilidad en el proceso experimental que favorece la extrapolación de las datos hacia casos más generales. Sin embargo, y antes de que se pueda llegar a ninguna conclusión, es preciso continuar con los estudios siguientes.

Dados todo los resultados obtenidos por las técnicas, es posible calcular el error de precisión asociado a cada una de ellas. De esta manera, se obtendrá un porcentaje indicativo de cuánto difiere el valor estimado del real en cada punto. En el caso de que este error sea negativo, indicará que el valor ha sido subestimado, y cuando sea positivo, que ha sido sobrestimado.

Se ha calculado la media de los errores (expresados en %) asociados a los puntos de cada grafo, resultados estos que se pueden observar en la Tabla 7.1. Se ha optado por esta representación, en lugar de una gráfica, debido a la mayor accesibilidad de los datos ofrecidos de esta forma.

La primera conclusión directa que se extrae es que el error asociado a la técnica sin reuso es muy elevado, tanto en términos absolutos como en comparación con los del resto. Este error es del orden del 150%, lo cual descalifica inmediatamente a esta técnica, hecho este que ya se podía presentir tras los párrafos anteriores.

La precisión de las otras metodologías, es sin embargo muy aceptable, con rangos de error justificables teniendo en cuenta la no consideración de detalles importantes, como el coste de las interconexiones.

Grafo	Plana (%)	Escalar (%)	Vectorial (%)	Sin Reuso (%)
1	18.0679	11.0365	8.6220	107.1948
2	27.0496	13.9938	15.3134	144.4269
3	25.7258	13.2317	6.9801	187.7104
4	22.3515	7.0452	11.1356	185.9498
5	14.6448	10.2960	11.0904	189.3589
6	21.2896	12.1086	18.1472	190.3996
7	13.6205	11.9090	13.5971	132.3363
8	16.0289	15.3297	13.3863	145.2828
9	14.4847	14.1711	23.0809	156.5638
10	14.2723	9.2325	3.6878	145.7333
11	10.9569	5.4426	9.6127	111.9849
12	17.8972	17.4394	17.3404	200.2927
13	8.4999	6.4480	8.0001	119.3712
14	16.6536	13.3998	7.9077	134.9297
15	12.0652	9.0465	18.1490	192.4244
16	18.6250	18.5346	21.3276	164.9665
17	12.9440	12.8343	15.3087	143.2847
18	15.0387	11.6063	21.6522	184.8020

Tabla 7.1

Resultados del error de precisión.

Se puede observar que la técnica escalar siempre ofrece un error menor que la plana. Sin embargo, esta generalización no se puede extender a la vectorial, ya que a veces llega a resultados más positivos, y a veces comete un error mayor.

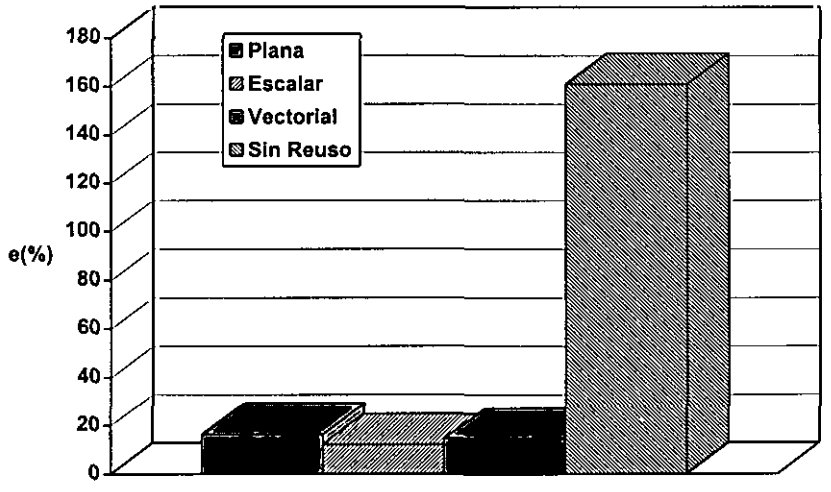


Figura 7.11

Media del error de precisión

En la Figura 7.11, se detalla la media total de los resultados obtenidos en los 204 experimentos[4], con el fin de sintetizar aún más la información. Se observa que las tendencias extraídas anteriormente se siguen manteniendo: la técnica plana ofrece un error del 16.31%; la escalar del 12.12%, por lo que se consiguen mejores resultados; y la vectorial del 14.44%, que, aunque se vio previamente que a veces obtenía valores más ajustados, otras veces realizaba una peor estimación, que es la que domina en la media general.

Este último dato va en contra de lo previsible en una primera instancia, ya que la utilización de una técnica más detallada podía inducir a pensar

[4] Aunque pueda parecer lo contrario, la media de los 204 experimentos no es equivalente a calcular la media agrupada por grafos, y realizar de nuevo la media sobre estos 18 grafos. Por eso, es normal que la media de los datos de la Tabla 7.1 no se corresponda con los ofrecidos a continuación.

Grafo	Plana (%)		Escalar (%)		Vectorial (%)		Sin Reuso (%)	
	Fidelidad	Error	Fidelidad	Error	Fidelidad	Error	Fidelidad	Error
1	90.00	0.2650	90.00	0.2650	90.00	0.2650	80.00	2.9232
2	100.0	0.0	96.42	8.4522	96.42	10.2071	89.28	31.5706
3	96.42	2.8478	96.42	2.8478	96.42	2.8478	85.71	14.3702
4	100.0	0.0	97.77	23.4051	93.33	23.5137	82.22	28.2620
5	98.18	0.9767	100.0	0.0	96.36	0.5215	87.27	29.6743
6	97.43	3.8431	96.15	5.0372	94.87	10.3760	91.02	11.4940
7	93.33	11.0066	95.55	11.5919	91.11	14.2304	84.44	37.9844
8	100.0	0.0	100.0	0.0	100.0	0.0	83.63	21.9602
9	96.36	19.9605	98.18	13.6758	96.36	19.9605	90.90	26.9962
10	100.0	0.0	98.18	5.0451	100.0	0.0	89.09	11.2582
11	100.0	0.0	100.0	0.0	100.0	0.0	91.67	29.4614
12	97.80	2.8329	95.60	12.3085	97.80	2.8329	91.20	13.4132
13	100.0	0.0	100.0	0.0	100.0	0.0	89.89	21.3385
14	93.33	4.3389	93.33	4.3389	95.56	3.9276	86.67	33.4101
15	93.94	17.9000	96.97	7.8031	95.45	13.9169	87.88	21.3791
16	95.24	6.9956	99.05	1.6126	97.14	8.7642	89.52	14.4118
17	96.32	4.4949	95.59	6.2774	95.59	8.6407	91.17	9.5841
18	97.79	5.8566	94.85	10.5806	93.38	8.2827	88.23	19.3306

Tabla 7.2

Resultados de la fidelidad

acerca de la obtención de mejores resultados. Sin embargo, esta previsible mejora se ve anulada por otros empeoramientos debidos a la no tan buena adecuación de esta técnica a los mecanismos macroscópicos.

De esta manera, al considerar funciones de densidad para cada tipo de unidad funcional, se está dispersando en exceso la información, con la consiguiente disminución del grado de abstracción, que redunda en problemas como el mal ajuste de los costes mediante curvas. Con esto, se demuestra como falsa la creencia tradicional de que la consideración de un número elevado de detalles es un hecho positivo.

Por otra parte, el error cometido por la técnica sin reuso es del 161.05%, lo cual está más allá de todo margen razonable. Aunque se pueda aducir que esta técnica es muy simple y rudimentaria, y que por lo tanto es normal que obtenga este tipo de resultados, hay que decir que todavía es usada por mucho sistemas, con la excusa de una simplicidad mal entendida.

Una vez concluido el estudio de la precisión, el siguiente punto trata la fidelidad de las técnicas de estimación. De nuevo, se ofrecen los resultados (en %) agrupados por grafos, esta vez en la Tabla 7.2. Una fidelidad del 100% indicaría que la técnica de estimación acertaría siempre en decidir la calidad relativa de dos soluciones.

Asimismo, en la misma tabla, se ofrece la media de resultados del error asociado a la fidelidad. Este error indica cuán peor es la solución que el estimador predice como mejor, respecto de la que realmente lo es. De esta manera, no es lo mismo equivocarse entre un par de soluciones que son muy próximas en calidad, ya que el error no va a tener una gran repercusión, que hacerlo con dos soluciones donde una es substancialmente mejor que la otra.

Tras el estudio de estos datos, se puede concluir que las técnicas plana, escalar y vectorial tienen una fidelidad muy alta y similar, por encima del 95%. En este caso, no es posible decidir qué técnica es mejor, ya que todas ofrecen unos resultados satisfactorios. El hecho de que las tres técnicas ofrezcan una fidelidad semejante, indica que todas tienen una tendencia paralela, pero es la escalar la que discurre de una forma más

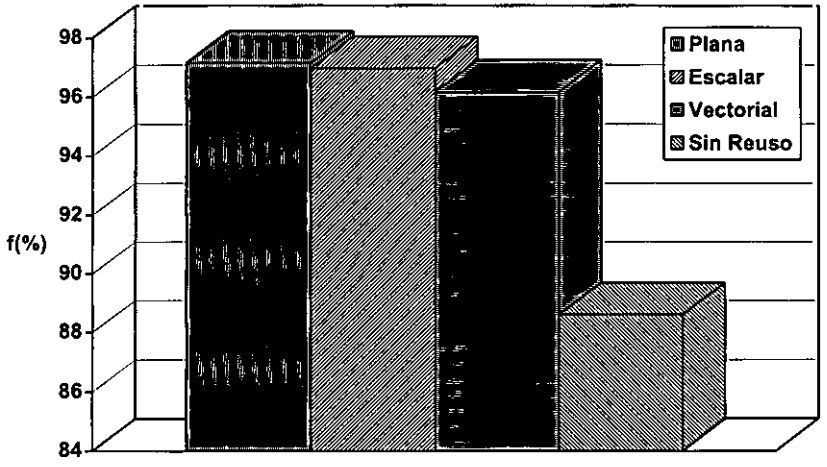


Figura 7.12

Media de la fidelidad

ajustada al valor real, ya que su error de precisión es ligeramente menor que el de las otras.

En cambio, y siguiendo con lo obtenido hasta ahora, la técnica sin reuso, ofrece una fidelidad inferior al 90%. Esto apoya las suposiciones previas, e inhabilita aún más a esta metodología para su empleo en problemas reales.

En el estudio del error de fidelidad, se observa que este valor es similar en el caso de las tres técnicas propuestas, y que está situado cerca del 10%.

El resumen detallado de la fidelidad se ofrece en la Figura 7.12. Como conclusión, se obtiene que este parámetro relativo a la aproximación plana es de un 97.15%; de un 96.97% respecto a la escalar; de un 96.17% respecto a la vectorial; y de un 88.61% respecto a la técnica sin reuso. Estos resultados corroboran las predicciones realizadas en los párrafos anteriores.

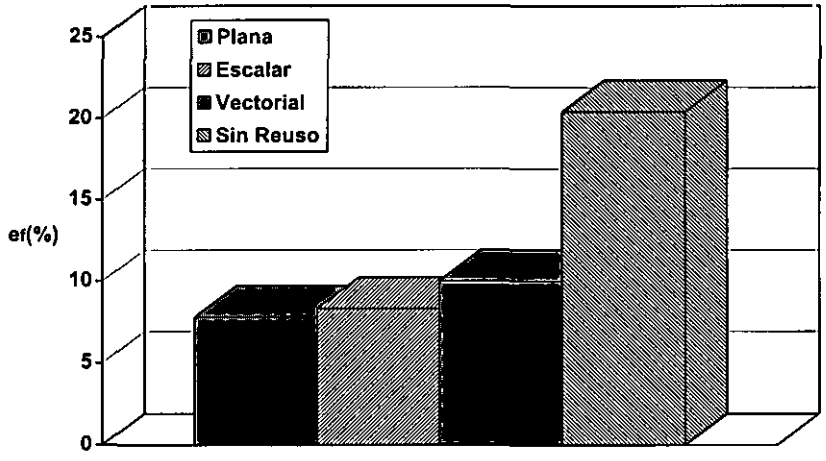


Figura 7.13

Media del error asociado a la fidelidad

Igualmente, la media general del error asociado a la fidelidad se muestra en la Figura 7.13. De esta manera, se obtiene que para la técnica plana el error es del 7.81%; del 8.32% para la escalar; del 10.03% para la vectorial; y del 20.44% para la técnica sin reuso.

Como conclusión, afirmar que las tres técnicas propuestas poseen una buena fidelidad, que logra mejorar a la proporcionada por el caso trivial en el que no se considera el reuso.

El último estudio de esta sección está dedicado al tiempo de obtención de los resultados mediante las distintas técnicas, que como se comentó anteriormente constituye una de las características básicas dentro del proceso de particionamiento.

De esta manera, se comparará este valor asociado a cada una de las tres técnicas propuestas con el equivalente al consumido por el *Behavioral Compiler* para medir el valor real. Se ha obviado en este caso el tiempo

Grafo	Plana (s)	Escalar (s)	Vectorial (s)	Sin Reuso (s)
1	0.0011	0.0532	0.2468	284.8000
2	0.0014	0.0418	0.5711	647.7500
3	0.0017	0.0705	0.6750	1124.2500
4	0.0021	0.0397	0.8613	266.0000
5	0.0028	0.1091	0.8269	883.7272
6	0.0031	0.0986	0.8915	747.0769
7	0.0011	0.0413	0.2080	203.7000
8	0.0013	0.0828	0.3312	289.0909
9	0.0017	0.2375	0.6396	1162.3636
10	0.0021	0.2609	1.1297	564.0000
11	0.0021	0.1820	0.8427	465.6667
12	0.0031	0.1530	0.6010	886.1429
13	0.0012	0.0481	0.3012	250.0000
14	0.0013	0.0487	0.2388	371.3000
15	0.0018	0.2243	0.9096	375.5000
16	0.0021	0.1673	0.8939	682.5333
17	0.0025	0.1982	0.4461	495.8823
18	0.0029	0.2325	1.3329	716.3529

Tabla 7.3

Resultados del tiempo de proceso

relativo a la técnica sin reuso, ya que la operación asociada es una simple suma. Obviamente, el tiempo dedicado a este cálculo es despreciable, virtud esta que se queda sin valor tras los malos resultados obtenidos por esta técnica en el caso de la precisión y la fidelidad. En la Tabla 7.3 se ofrece el resumen de los tiempos de cálculo (en segundos) para las metodologías comentadas.

Se observa que existe una gran diferencia de magnitud entre unas y otras. Así, se ve que el tiempo de la técnica plana es del orden de las milésimas de segundo; el de la escalar del orden de las centésimas; y el de la vectorial del orden de las décimas. Sin embargo, para el cálculo del valor *real se han obtenidos valores de tiempo del orden de centenares de* segundos, y que en algunos casos rebasan el millar.

Estas diferencias de tiempo son cruciales en Codiseño. Las técnicas propuestas son perfectamente integrables dentro de un entorno de particionamiento, no así el cálculo del valor verdadero, que haría el proceso totalmente inviable[5]. Nótese el decremento tan grande de tiempo producido, manteniendo unos niveles de calidad, dados por la precisión y la fidelidad, muy aceptables.

El resumen general de tiempos se ofrece en la Figura 7.14. No se ha representado el dato asociado al valor real, debido a la gran diferencia de su orden de magnitud, que no dejaba contemplar el resto de las aproximaciones con detalle.

De esta manera, se obtiene que el tiempo medio para la técnica plana es de 0.0021 segundos; de 0.1388 para la escalar; de 0.6993 para la vectorial; y de 593.1576 para la medición del valor real.

Un detalle significativo es que la técnica vectorial tarda unas cinco veces más que la escalar, dato este muy razonable, si se tiene en cuenta que la biblioteca usada contiene cinco unidades funcionales. Así, en este caso habría que repetir cinco veces todos los cálculos realizados para la aproximación escalar.

[5] Aquí se entenderá mejor lo comentado en §3, acerca de los problemas de tiempo en Codiseño, y la necesidad de nuevas técnicas de estimación, a no ser que se utilicen problemas simplificados en exceso.

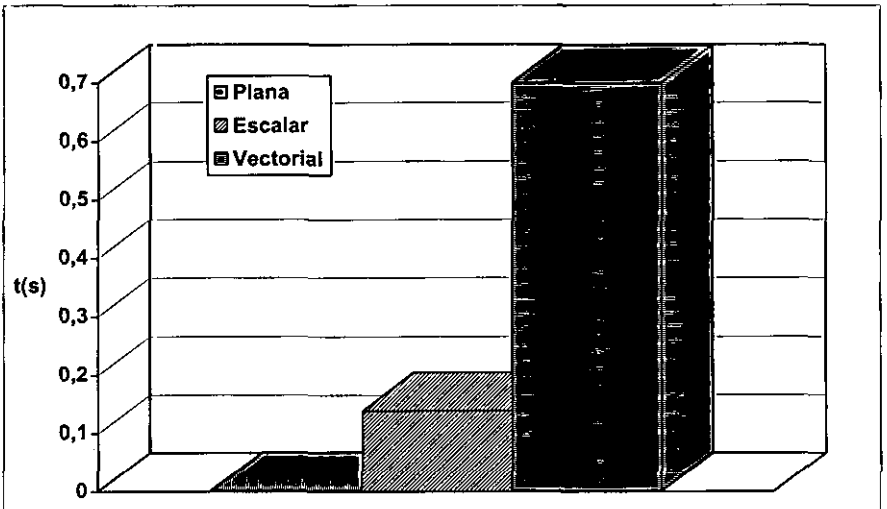


Figura 7.14

Media del tiempo de proceso

Con esto acaba el estudio destinado a las técnicas de estimación propuestas, y a su comparación con los casos extremos de no consideración de reuso, y de medición de los valores reales.

7.2.4. Conclusiones acerca del proceso experimental.

Tras la realización de las pruebas experimentales comentadas en la sección anterior, es posible realizar una extrapolación de las conclusiones extraídas, que catalogue las características introducidas por las técnicas presentadas. Esto es posible debido a que el número de experimentos realizado es lo suficientemente amplio para procesos de este tipo. Este hecho lo corrobora la tendencia observada de regularidad en los experimentos, lo que proporciona una seguridad en cuanto a la buena realización de los mismos.

La primera conclusión que se puede deducir es el cumplimiento del objetivo que motivó el presente trabajo, que no es otro sino la reducción del tiempo de estimación con el fin de poder integrar el proceso en una metodología de Codiseño de una forma viable. Esto se ha cumplido con creces, si se comparan los tiempos requeridos por las nuevas técnicas con el empleado por la herramienta para calcular el valor real, la cual ofrece un resultado varios órdenes de magnitud mayor.

Esta gran diferencia es la que permitirá tratar problemas de una entidad altamente compleja, con garantías de que se puedan solventar de una forma razonable. Gracias a esto, ahora es posible descubrir la gran cantidad de tiempo que se ha empleado tradicionalmente en el tratamiento de detalles internos a los nodos, que si bien conducen a una solución precisa, pierden parte de su trascendencia al ser tratados desde este nivel del trabajo.

No obstante, el hecho de la reducción del tiempo de proceso por sí solo no constituye un ventaja clara añadida. Por poner un caso, la técnica de estimación trivial sin la contemplación de reuso es prácticamente instantánea, y sin embargo queda altamente descalificada por los malos resultados que proporciona, desviándose totalmente de los valores reales.

De esta manera, la eficiencia sin calidad no es útil en estos tipos de metodologías. Sin embargo, las técnicas presentadas conjugan estas dos virtudes, ya que los márgenes de precisión y fidelidad son muy satisfactorios, tratándose de un modelo con un gran nivel de abstracción. Errores de precisión inferiores al 15%, con una fidelidad superior al 95% son totalmente admisibles dentro de un entorno de particionamiento como es el de Codiseño.

La comparación de estas tres técnicas entre sí, ofrece la conclusión de que la precisión de la técnica plana se ve superada por la escalar y la vectorial, en unos pocos puntos porcentuales, debido a la consideración de

la estructura interna de las tareas, lo que añade nuevos criterios a la hora de tratar el posible reuso hardware.

Sin embargo, la técnica vectorial no consigue mejorar claramente a la escalar. En la primera, se observa una oscilación en los resultados, por la que a veces se sobrestiman y a veces se subestiman los valores reales, siendo la media general algo peor que la segunda. Así, la consideración de una única función densidad tiene más sentido, en un nivel macroscópico, que el trabajar con múltiples densidades asociadas a las distintas unidades funcionales[6].

Respecto a la fidelidad, hay que decir que todas las técnicas cumplen esta propiedad de modo notable. El hecho de aumentar en ellas la precisión, no ha disminuido la tendencia equilibrada que sigue la evolución de los valores reales.

En consecuencia, las tres técnicas son convenientes para ser integradas en un entorno de particionamiento, ya que en esta situación, la consideración más importante viene dada por la relación de calidad entre soluciones.

En el caso de querer disponer de un método con mayor precisión para estudiar el cumplimiento de ciertas restricciones impuestas, sería interesante la aplicación de la técnica escalar. Estas conclusiones, que *a priori* no eran las esperadas, corroboran la consistencia del entorno macroscópico.

Esta consistencia se vislumbró previamente en el desarrollo matemático que ha sustentado toda la teoría. Sin embargo, es ahora, al considerar el conjunto de pruebas experimentales, cuando se obtiene una mayor seguridad en este aspecto.

---

[6] Aunque esta percepción sea totalmente contraria de la que se obtendría en un entorno microscópico. Por eso, es conveniente tener sumo cuidado a la hora de realizar expansiones del conjunto macroscópico, para no traspasar la frontera entre los dos niveles.

Por todas las razones anteriores, creo que las innovaciones presentadas en el campo de la estimación son apropiadas y necesarias en un entorno de problemas con una complejidad elevada.

7.3 Resultados experimentales obtenidos para el proceso de particionamiento.

7.3.1. Valores de contorno utilizados en el proceso.

Al igual que lo efectuado en el proceso experimental sobre las estimaciones, es preciso especificar los detalles que rodean esta prueba, con el fin de centrarla entre las distintas posibilidades que se presentan.

Primeramente, el banco de diseños generados vuelve a ser de dieciocho, como en el caso anterior, aunque estos grafos han sido creados aleatoriamente de nuevo, y por lo tanto son distintos a los usados en las pruebas de estimación. También aquí se ha optado por tres series de grafos, con un número de nodos comprendido entre 7 y 12.

El esquema de parámetros utilizado sobre el generador aleatorio de grafos también ha sido el mismo, tanto acerca de la composición de las tareas como de las características de las unidades funcionales. De esta manera, aunque los grafos sean diferentes a los anteriores, siguen siendo isomorfos.

Para cada uno de los grafos, se han elegido entre 75 y 150 valores asociados de restricción temporal, lo que ha dado lugar a otros tantos procesos de particionamiento.

Cada uno de estos procesos ha sido realizado varias veces, utilizando distintas técnicas. Por un lado, se ha empleado el algoritmo estándar de *Fiduccia-Mattheyses*, y por otra parte, las técnicas de agrupamiento previas, tanto en el caso de una única etapa como en el de multi-etapa.

Como se ha venido comentando a lo largo de este trabajo, es fundamental la interacción que el proceso de particionamiento ha de realizar con la técnica de estimación de grafo asociada, ya que es esta la que proporciona los valores de los parámetros relacionados con las soluciones intermedias, que sirven para optar por la situación más conveniente en cada momento.

Por lo tanto, y a la vista de los resultados anteriores, se ha elegido la técnica de estimación plana para formar parte del actual proceso experimental.

Aunque las tres aproximaciones presentadas poseen una fidelidad parecida, factor este crucial en el proceso de particionamiento, se ha preferido optar por la plana[7], ya que el tiempo que consume en la producción de resultados es mucho menor que el asociado a las otras dos. De esta manera, se está acelerando el proceso de particionamiento de una forma notable.

Realmente, y en cuestión de la calidad de los resultados, no habría habido diferencia de haber elegido utilizar cualquiera de las tres técnicas presentadas. Eso es debido a su muy igualada fidelidad, ya que de esta manera, las diferencias en este aspecto, que únicamente rondan el 1%, no son suficientes como para hacer que el sistema se desvíe de la solución prevista.

[7] Si bien puede parecer que la fidelidad es el único factor trascendente en el proceso, sin tener en cuenta la precisión obtenida, esto no es así. De hecho, la única manera de obtener una idea del coste *real* del circuito es mediante una buena precisión, y en el caso de imponer restricciones sobre este coste en el proceso de particionamiento, la utilización de técnicas como la escalar, sería de vital importancia.

Respecto a los parámetros propios de la técnica de selección del criterio de agrupamiento, se han utilizado los mismos que en el ejemplo ofrecido en §6.4. Así, se ha tomado c_1=c_2=1, y h=0.9.

7.3.2. Estudios prácticos realizados.

Los estudios llevados a cabo en este módulo experimental tienen como finalidad la comparación de la técnica de particionamiento estándar con respecto a la metodología de agrupamiento previa. Esta comparación está orientada a comprobar la mejora de la *calidad* obtenida, con respecto al *tiempo de proceso* utilizado.

Asimismo, se pretende ver cómo estos factores cambian cuando el agrupamiento se realiza mediante la técnica multi-etapa. De esta manera, se contemplarán los casos de una única iteración (it=1), de un medio del número de iteraciones máximo (it=½Max), de tres cuartos del número de iteraciones máximo (it=¾Max) y del número de iteraciones máximo (it=Max).

a) *Calidad.* La calidad viene dada por el número de soluciones mejores y peores que se obtienen con una de las técnicas de agrupamiento, respecto al uso del algoritmo estándar. De esta manera, si el número de soluciones mejores es mucho mayor que el de peores, la calidad de ese proceso de agrupamiento será alta. Si, por el contrario, las soluciones peores predominan, la calidad será baja. Así, se ha realizado este estudio comparativo para cada una de las cuatro aproximaciones al agrupamiento mencionadas.

Por otra parte, no sólo es importante el número de soluciones que se ha logrado mejorar, sino por cuánto. De esta manera, si una técnica de agrupamiento encuentra un número de soluciones mejores, con un ahorro de coste considerable, y por otra parte halla un número igual de soluciones peores, pero con una penalización inapreciable, se dirá que dicha técnica resulta ventajosa. En consecuencia, se ha estudiado un factor de impacto

que mide este efecto, y que se calcula como el cociente entre el coste extra introducido por las soluciones peores, y el coste ahorrado gracias a las soluciones mejores. Cuanto más próximo a cero sea este factor, mayor calidad tendrá la técnica. De esta manera, se tiene que el factor de impacto de una técnica de agrupamiento dada es:

$$i_{gr} = \frac{\sum_{j \in P}(c_j^{st} - c_j^{gr})}{\sum_{k \in M}(c_k^{gr} - c_k^{st})} \qquad [7.2]$$

En esta expresión, c_j^{st} representa el coste de la técnica estándar para el diseño *j*, c_j^{gr} el equivalente para el caso de agrupamiento, *P* es el conjunto de soluciones hallado por la técnica de agrupamiento que son peores que en el caso estándar, y *M* el conjunto equivalente de soluciones mejores.

b) *Tiempo de proceso.* El tiempo consumido en producir los resultados es un factor muy importante. Así, las técnicas de agrupamiento que emplean más iteraciones supuestamente encontrarán un amplio rango de soluciones mejores, pero a costa de aumentar el tiempo requerido para ello. De esta manera, se ha realizado un estudio comparativo de estos tiempos para el particionamiento estándar y los múltiples procesos de agrupamiento previo.

7.3.3. Relación de los resultados obtenidos.

Primeramente, se ha realizado el proceso de particionamiento sobre cada uno de los 18 grafos considerados, en un amplio rango de restricciones temporales, comprendidas entre el tiempo asociado a la solución en que todos los nodos se encuentran en software (máximo), y el equivalente al caso en que todos ellos han sido asignados a hardware, con su implementación más rápida (mínimo).

Después, se ha procedido a la comparación de las soluciones producidas entre el algoritmo estándar y las distintas aproximaciones al agrupamiento. De esta manera, se han contabilizado el porcentaje de soluciones mejores halladas mediante esta metodología, así como el porcentaje de soluciones peores y de soluciones iguales.

Se pretende comprobar si la incidencia de la innovación introducida resulta positiva en el contexto general del particionamiento. En la Tabla 7.4, aparecen los resultados de este estudio. En este caso, ha resultado preferible la utilización de una tabla, en vez de una gráfica, con el fin de condensar la gran cantidad de información disponible.

De esta manera, se explicitan en cada caso los porcentajes de soluciones mejores halladas por el algoritmo estándar, *S%* (donde el agrupamiento no es eficaz); los porcentajes de soluciones mejores halladas por la técnica de agrupamiento correspondiente, *C%* (donde esta metodología sí ha sido conveniente); y el porcentaje de soluciones iguales halladas en ambos casos, *I%*.

Un primer análisis de estos resultados, ofrece la conclusión de que mediante la realización previa de un proceso de agrupamiento, se obtienen un gran número de soluciones mejores, que habían sido ignoradas por el algoritmo de particionamiento estándar. Esto es debido a una mala exploración del espacio de búsqueda, influida por la presencia negativa de altas comunicaciones en el sistema.

Se observan algunos casos en los que, sin embargo, la técnica de agrupamiento ofrece un porcentaje significativo de soluciones peores. Esto, como se comprobará luego, no tiene una incidencia excesiva, debido a que el error cometido en estos casos no lleva asociada una alta penalización de coste.

Este efecto se ve producido por el hecho de que al ajustar mucho los criterios de agrupamiento, es posible considerar alguna comunicación

#	It=1			It=½Max			It=¾Max			It=Max		
	S%	C%	I%	S%	C%	I%	S%	C%	I%	S%	C%	I%
1	0.00	33.85	66.15	0.00	33.85	66.15	0.00	33.85	66.15	0.00	33.85	66.15
2	13.72	52.57	33.71	13.14	52.57	34.29	13.14	52.57	34.29	0.00	52.57	47.73
3	20.64	76.98	2.38	20.64	79.68	2.38	0.79	76.98	22.23	0.00	76.98	23.02
4	48.34	28.33	23.33	48.34	28.33	23.33	18.33	30.00	51.67	0.00	30.00	70.00
5	26.23	72.13	1.64	22.95	72.13	4.92	0.00	72.13	27.87	0.00	72.13	27.87
6	6.15	61.54	32.31	6.15	61.54	32.31	6.15	61.54	32.31	0.00	61.54	38.46
7	1.33	48.00	50.67	1.33	48.00	50.67	1.33	48.00	50.67	0.00	48.00	52.00
8	0.00	34.55	65.45	0.00	34.55	65.45	0.00	34.55	65.45	0.00	34.55	65.45
9	11.54	25.64	62.82	6.41	25.64	67.95	0.00	25.64	74.36	0.00	25.64	74.36
10	46.75	45.45	7.80	45.46	46.75	7.79	1.30	58.44	40.26	0.00	58.44	41.56
11	1.45	55.07	43.38	1.45	55.07	43.48	1.45	55.07	43.48	0.00	55.07	44.93
12	0.00	63.49	36.51	0.00	63.49	36.51	0.00	63.49	36.51	0.00	63.49	36.51
13	37.27	23.64	39.09	5.45	23.64	70.91	0.00	23.64	76.36	0.00	23.64	76.36
14	0.00	39.39	60.61	0.00	39.39	60.61	0.00	39.39	60.61	0.00	39.39	60.61
15	10.13	30.38	59.49	6.33	30.38	63.29	0.00	30.38	69.62	0.00	30.38	69.62
16	36.43	28.85	34.62	36.43	28.85	34.62	9.61	28.85	61.54	0.00	28.85	71.15
17	14.75	49.18	36.07	0.00	57.38	42.62	0.00	57.38	42.62	0.00	57.38	42.62
18	0.00	13.11	86.89	0.00	13.11	86.89	0.00	13.11	86.89	0.00	13.11	86.89

Tabla 7.4

Resultados del proceso de particionamiento

ambigua en la frontera entre la necesidad de evitarla y la de agruparla. Pero estos casos, en los que dichas comunicaciones no son muy trascendentes, no suponen la obtención de diseños esencialmente diferentes, por lo que equivocarse aquí no trae consigo consecuencias excesivamente graves.

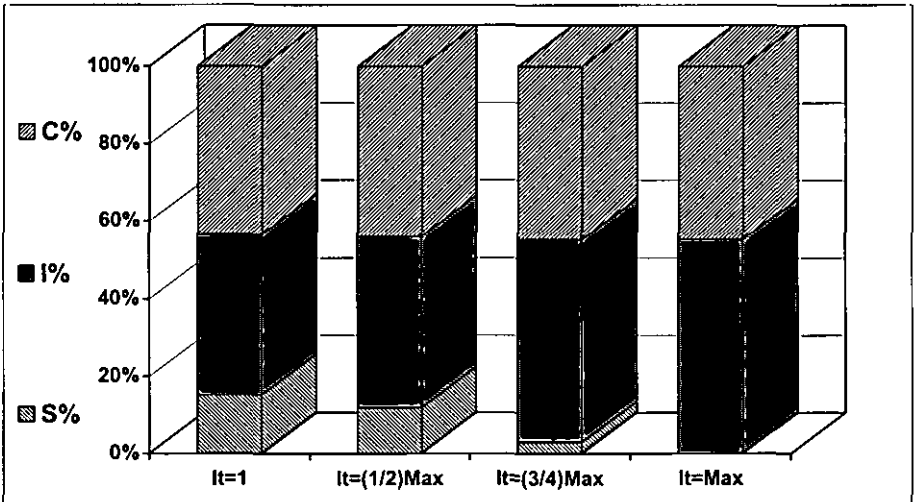


Figura 7.15

Media comparativa de los resultados de particionamiento

De esta manera, y como podría resultar obvio *a priori*, a medida que se consideran procesos de agrupamiento con más iteraciones, el porcentaje de soluciones peores obtenidas va disminuyendo progresivamente.

Otro dato también claro, es que en el último caso, donde las comunicaciones agrupadas son liberadas hasta obtener el espacio inicial formado por nodos individuales, este porcentaje de desventaja se hace nulo, y todas las soluciones obtenidas son mejores o iguales.

Se ha realizado la media global de los resultados anteriores, con el fin de llegar a una conclusión general. Este resumen se observa en la Figura 7.15. Aquí, se ve la evolución de los distintos porcentajes de soluciones mejores, iguales y peores, al realizar un número progresivo de iteraciones.

En el primer caso, donde únicamente se realiza la iteración básica, ya se obtienen resultados muy favorables si se lleva a cabo el agrupamiento previo (*C%*=43.46%, *I%*=41.28%, *S%*=15.26%). Estos datos indican el gran

porcentaje de soluciones que eran totalmente ignoradas por el algoritmo estándar.

A continuación, cuando se realiza un número de iteraciones suficiente como para liberar la mitad de las comunicaciones agrupadas, se observa que el porcentaje de soluciones peores se ha logrado disminuir (*C*%=43.99%, *I*%=44.12%, *S*%=11.89%).

Si se sigue realizando este proceso, y se llega a un número de iteraciones que libere las tres cuartas partes de las comunicaciones inicialmente agrupadas, se ve que este porcentaje de soluciones peores ha pasado a ser insignificante (*C*%=44.72%, *I*%=52.38%, *S*%=2.90%).

Finalmente, si se siguen liberando comunicaciones, hasta que no quede ninguna agrupada, se observa que la totalidad de las soluciones encontradas son mejores o iguales que las halladas por el algoritmo estándar (*C*%=44.72%, *I*%=55.28%, *S*%=0.00%).

De esta manera, se comprueba que la realización de un proceso de agrupamiento previo es una propuesta razonable, no sólo porque obtenga resultados mejores que el particionamiento clásico, sino por las serias deficiencias presentadas por este en la búsqueda de soluciones óptimas.

Existe un dato importante que se puede extraer de la gráfica anterior. Este dato está relacionado con la evolución que sufren los distintos tipos de soluciones. Se observa, que sólo con el caso inicial de una única iteración, se han encontrado la mayoría de las soluciones mejores que el algoritmo estándar no fue capaz de hallar (un 43.46% frente a un 44.72% en el caso límite). De esta manera, el criterio de agrupamiento propuesto resulta muy eficaz.

Así, la técnica multi-etapa aligera esta rigidez, dejando evolucionar libremente a las comunicaciones intermedias, lo que permite un cierto progreso del sistema, que ahora sí es capaz de encontrar la solución hallada por el particionamiento clásico. De esta manera, se observa que el

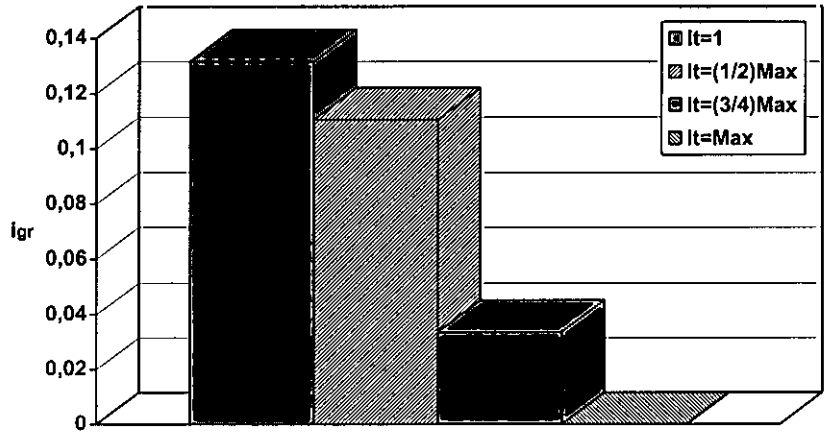


Figura 7.16

Media del índice de impacto

porcentaje inicial de soluciones peores, un 15.26%, se ve absorbido casi en su totalidad por el porcentaje de soluciones iguales.

Realizados esta serie de comentarios, es preciso proceder ahora con el estudio del índice de impacto de las soluciones peores encontradas. De esta manera, aunque estas constituyan un número elevado, si el coste extra que conllevan es reducido, su influencia se ve altamente disminuida. En consecuencia, se aplica la expresión del índice dada por [7.2] sobre los resultados anteriores, obteniendo el resumen general ofrecido en la Figura 7.16.

Se ve que incluso en la aproximación con una única iteración, la relación entre los costes de las soluciones peores encontradas respecto a los costes de las mejores es muy baja. De esta manera, este índice va disminuyendo comenzando en 0.1316, pasando por 0.1106 y 0.0338 hasta llegar a ser nulo.

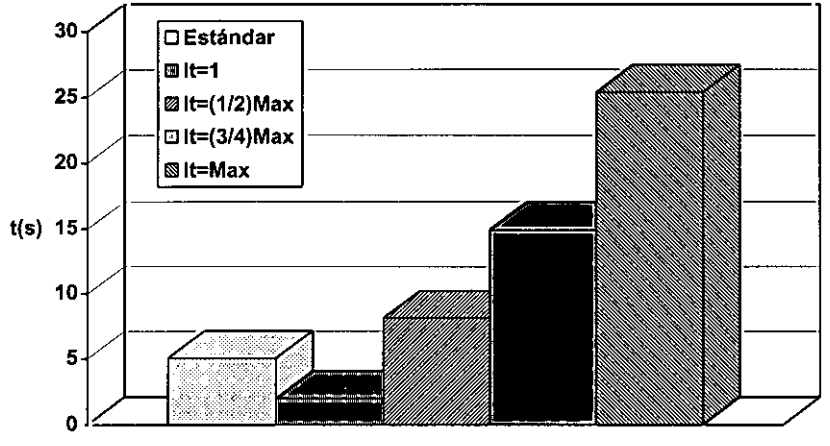


Figura 7.17

Media del tiempo de proceso

La conclusión de estos datos es que en los casos en los que el proceso de agrupamiento encuentra soluciones peores, estas no son realmente significativas, y acaban siendo compensadas por los casos en las que las soluciones halladas son mejores.

El siguiente estudio realizado gira alrededor del tiempo consumido por cada una de las técnicas para obtener los resultados finales. De esta manera, se ofrece en la Figura 7.17 la media de estos tiempos (en segundos) relacionados con el total de procesos de particionamiento llevados a cabo sobre el conjunto de grafos.

Se puede observar que el algoritmo estándar requiere una media de 5.04 segundos para efectuar un cálculo. Sin embargo, la aproximación de agrupamiento con una única iteración, reduce este tiempo a una media de 2.01 segundos. Esto es debido al hecho de que al aplicar esta técnica, los distintos nodos se colapsan en la formación de grupos, disminuyendo por tanto el número de elementos tratados.

Como el tiempo de particionamiento es proporcional al número de objetos que se tienen que manipular, este tiempo se ve reducido respecto al caso estándar. Así, el agrupamiento no sólo redunda en la obtención de mejores resultados, sino que también se ve mejorada la eficiencia del proceso.

Estos tiempos, a medida que se van realizando más iteraciones, sufren un incremento, como es lógico, evolucionando hacia los 8.13 segundos, los 14.96 segundos y los 25.39 segundos de media respectivamente. Este aumento de tiempo es consecuencia de una mayor exploración del espacio de diseño.

Con esto, finalizan los estudios realizados sobre los resultados experimentales obtenidos en el proceso de particionamiento.

7.3.4. Conclusiones acerca del proceso experimental.

A continuación, es necesaria una extrapolación de las conclusiones que se derivan de los valores hallados en la sección anterior. La primera reflexión es relativa a la veracidad de las suposiciones vertidas en §6.1.2, acerca de la distorsión introducida por las comunicaciones del sistema en el espacio de búsqueda, y a la imposibilidad de los algoritmos estándar de solventar esta situación.

Como se ha visto, el número de casos en los que la utilización del método clásico llevaba asociado un considerable empeoramiento de la solución obtenida es muy elevado, perdiendo mucha de la eficacia del proceso.

Un hecho colateral a este es que dicha distorsión se produce aún contando con un mecanismo de estimación muy ajustado, ya que el problema no proviene de este campo, sino que es algo intrínseco a la estructura de los diseños utilizados.

De esta manera, todo el esfuerzo destinado a la mejora de los estimadores se ve desperdiciada si no se solventa el problema del particionamiento de una forma eficaz.

La técnica de agrupamiento soluciona en gran medida esta desventaja. Incluso en el caso de una única iteración, los resultados obtenidos son muy beneficiosos, comparados con los del caso clásico. Esto reafirma la idea de que la selección del criterio de agrupamiento está bien constituida, y que es apropiada para su integración en este tipo de sistemas.

Otra conclusión que se puede extraer es que llevando a cabo la aproximación de agrupamiento con un número máximo de iteraciones, los resultados obtenidos son siempre mejores, y de esta manera se conjugan las ventajas de las dos técnicas utilizadas: el algoritmo estándar y el proceso de agrupamiento previo.

No obstante, y tras la observación de los resultados, es posible concluir que no es necesario acudir hasta el caso límite para la obtención de unos valores correctos. De esta manera, y si se realiza el proceso multi-etapa con un número de iteraciones aproximadamente igual a tres cuartos del máximo, se comprueba que los resultados hallados son muy positivos en la comparación de calidad y tiempo de proceso.

Así, se observa que en este caso, el porcentaje de soluciones peores que se ha hallado es insignificante, y que se puede estimar alrededor del 3%. Este valor no tiene un peso consistente si se compara con el porcentaje de soluciones mejores halladas, cercano al 45%.

Por otra parte, el índice de impacto de estas soluciones peores también es muy reducido en el conjunto de la globalidad, y se sitúa algo por encima del 0.03. Esto, en otros términos, indicaría que las soluciones peores estarían introduciendo una carga extra de coste de un 3% respecto al coste ahorrado por el conjunto de soluciones mejores.

Las afirmaciones anteriores se ven corroboradas por el estudio de tiempos realizado. La técnica de agrupamiento mencionada tendría un tiempo de proceso medio de aproximadamente 15 segundos. Si se compara con dicho tiempo asociado a la simple ejecución del algoritmo estándar, que resulta ser algo superior a 5 segundos, se observa que el incremento producido compensa realmente a la vista de las mejoras obtenidas en la calidad.

De todas maneras, es el propio diseñador el que debe decidir dónde está la frontera razonable entre tiempo de exploración y calidad. Habrá casos en los que esta calidad sea un factor importante, y se deberá llegar hasta el límite máximo de iteración, y habrá situaciones en las que el tiempo sea más determinante, donde el proceso deberá ser detenido en etapas previas.

En resumen, por todos los datos presentados a lo largo de esta sección, el empleo de una técnica de agrupamiento previa al particionamiento está totalmente justificada, y la selección del criterio propuesta es acorde a las necesidades del problema.

Conclusiones y trabajo futuro.

8

Después de la presentación del trabajo experimental realizado, que ha servido para constatar las ventajas inherentes de la metodología ofrecida a lo largo de esta memoria, llega el momento de recapitular todas las conclusiones extraídas hasta ahora, y verificar si los objetivos marcados en un primer momento han sido cumplidos. De esta manera, resumiré en este capítulo los aspectos más destacados del presente trabajo, a fin de establecer el conjunto de características teóricas y prácticas tratadas.

Por otra parte, toda investigación quedaría incompleta sin una propuesta de continuación de ciertos puntos, que si bien han sido mencionados, no se han desarrollado con el nivel de profundidad de los que aparecen en capítulos previos. Así, en la última parte de este trabajo sugiero la posibilidad de expandir ciertos aspectos interesantes no tratados. De esta manera, pretendo que todo lo propuesto hasta ahora, que conforma una nueva metodología *per se*, sirva de plataforma de investigación para futuras extensiones del modelo, con el propósito de que finalmente se obtenga un sistema integrado, con capacidad de abordar los cada vez más complejos problemas de Codiseño.

8.1 Conclusiones acerca del trabajo presentado.

Las motivaciones que llevaron al planteamiento y posterior desarrollo de esta investigación fueron muy claras. De esta manera, se pretendía dar una visión nueva a las técnicas de aproximación al Codiseño, que si bien ofrecen buenos resultados al trabajar sobre problemas concretos, se muestran ineficaces al tratar de abordar los cada vez más complejos diseños existentes.

Así, se vio que las metodologías tradicionales heredadas de la Síntesis de Alto Nivel cometían el error de realizar una simple expansión de los algoritmos y técnicas ya conocidos, para ser utilizados con los nuevos problemas emergentes (§3.1).

Por lo tanto, y como respuesta a unas necesidades reales y actuales, se propuso no ya una mejora de las técnicas de estimación habituales, sino una nueva concepción global para el tratamiento de los problemas de Codiseño. Esta, se basa en la percepción clara de que si la complejidad de un campo de trabajo se incrementa hasta llegar a ciertos límites de tratabilidad, es preciso aumentar el grado de abstracción con que se abordan estos problemas (§3.2).

Por esta razón, opté por la consideración de un nuevo nivel catalogado bajo el término *macroscópico*. Esta innovación suponía la utilización de un nuevo conjunto de datos y métodos, distintos a los tradicionales, que permitiesen realizar las tareas de estimación en un tiempo mucho más razonable que el actualmente propuesto.

Así, si se respetaba la frontera entre niveles, y una vez comenzado el proceso de Codiseño se evitaban todos los detalles internos en la composición de las tareas, ciñiéndose exclusivamente a su traducción macroscópica, se abogaba por una mayor eficiencia de los algoritmos, que condujese a una más elevada tratabilidad de los problemas actuales.

De esta manera, se catalogaron una serie de datos macroscópicos, clasificándolos en intrínsecos y extrínsecos (§4.1). Entre los primeros, se encontraban el número de implementaciones hardware de cada nodo, el tiempo, el coste y la semejanza. Los segundos estaban caracterizados por el estado de los nodos y el solapamiento entre ellos.

Con estos datos, se propuso una estructura interna, capaz de albergar los distintos grados de paralelismo entre tareas y reuso hardware, de tal manera que el particionador pudiera llevar a cabo la determinación de estos dos factores trascendentes de una forma eficaz (§4.3).

Así, se vio que la estructura interna era tal que contenía toda la información de datos y control necesaria, y por lo tanto, mediante un sencillo cálculo del camino crítico del grafo asociado, era posible encontrar el tiempo de ejecución del sistema. De esta manera, se fijaba en un cierto grado el rango de movilidad de las distintas tareas (§4.4).

Una vez realizado esto, era preciso llevar a cabo un proceso de planificación macroscópica para determinar el instante exacto de comienzo de cada una de dichas tareas, antes de proceder con el cálculo del coste asociado.

Esta planificación se regía por el criterio de no coincidencia entre los pares de nodos con una gran semejanza (§4.5). Gracias a esto, se permitía una mayor probabilidad de reuso hardware, lo que redundaría en una disminución del coste final, y por lo tanto, en el aumento de la calidad de la solución. El proceso se lleva a cabo mediante el uso de un algoritmo basado en la planificación por listas, que va reduciendo la movilidad de los nodos hasta que estos quedan fijados en un instante en concreto.

Una vez realizada esta labor, se procedía con el cálculo del coste propiamente dicho (§4.6). Entonces, según criterios basados en el solapamiento y la semejanza de un par de nodos, se propuso una serie de

expresiones que calculaban la parte proporcional de una tarea dada que podía ser compartida con las otras funcionalidades existentes en hardware.

Gracias a esto, la caracterización del grafo de Codiseño se realizaba rápidamente, y los parámetros de tiempo y coste estaban disponibles para que el particionador decidiese acerca de la calidad relativa de la solución.

Con los datos macroscópicos y las técnicas de cálculo de tiempo de ejecución, planificación y cálculo de coste, el nivel de trabajo propuesto quedaba caracterizado. No obstante, surgió la cuestión acerca de si esta metodología podía ser completada o extendida, a fin de que reflejase más aspectos reales de los diseños.

Así, se propuso una expansión del conjunto de datos, para que sirviera como ejemplo a futuras innovaciones. El hecho fundamental, que ha debido quedar claro, es que mediante este proceso no se aboga por un cierto abandono de los datos macroscópicos, optando por la introducción de ciertos parámetros microscópicos. De hacer esto, se estaría violando la estancabilidad de los niveles, lo cual podría conllevar que los procesos no se comportasen tan favorablemente como los propuestos. Lo que se pretende, es la consideración de más aspectos, pero siempre desde el mismo grado de abstracción.

De esta manera, se propuso el problema de la heterogeneidad de la estructura de las tareas (§5.1). Debido a esta circunstancia, suposiciones globales acerca de la posibilidad de reuso hardware dejaban de cumplirse, dando paso a matizaciones más delicadas. Así, podía darse el caso de tareas que se ejecutaban en paralelo, pero por la particular distribución interna de sus unidades funcionales, que no producía ningún tipo de conflicto, la probabilidad de reuso era máxima. Sin embargo, existían casos en los que el solapamiento entre tareas era mínimo, pero que precisamente en ese punto de coincidencia era donde se requería el uso de las mismas unidades simultáneamente, privando de la posibilidad de compartirlas.

Con el fin de solucionar este problema, se propuso la caracterización de la estructura interna de las tareas mediante un nuevo parámetro macroscópico, denominado densidad (§5.2.1). Así, haciendo uso de una representación polinómica, se facilitaban las regiones donde existía una mayor carga operacional, y por lo tanto donde se podrían producir las máximas posibilidades de conflicto.

Esta innovación requería una nueva definición del concepto de solapamiento (§5.2.2), ya que en ese punto no solamente era importante el *tamaño de la región de coincidencia, sino la densidad operacional que* existía en ella. Esta definición estaba basada en el cálculo integral, lo que producía operaciones muy sencillas, debido al fácil tratamiento de las funciones polinómicas.

De esta manera, se comprobó que esta consideración de la distribución interna de las operaciones era positiva, obteniendo mejores resultados que la técnica original, denominada técnica *plana*.

A continuación, se hizo un estudio ulterior acerca del porqué de haber tratado una única función de densidad para cada tarea, y no una función de densidad separada para cada uno de los distintos tipos de unidades funcionales existentes (§5.3.1).

Así, se procedió a realizar una expansión de datos diferente a la anterior, ahora con la consideración de estas múltiples funciones de densidad. Se propuso un desarrollo teórico, en el cual, partiendo de suposiciones iniciales distintas a las de los casos previos, se llegó a conclusiones similares, dando coherencia a magnitudes *a priori* artificiales como la semejanza (§5.3.2).

Esto demostraba la consistencia del conjunto macroscópico, y la buena realización de las expansiones presentadas. Sin embargo, y al probar la última técnica introducida, denominada *vectorial* en contraposición con la

previa (*escalar*), se observó que los resultados obtenidos no mejoraban los anteriores de la manera que se esperaba (§5.3.3).

La explicación a este fenómeno radicaba en que si bien se admitían más aspectos, estas características no eran *tan* trascendentes en el grado de abstracción actual como se podía creer en un principio. Así, la ventaja que se adquiría mediante la separación de las distintas unidades funcionales se compensaba con otros factores negativos, como el mal ajuste de las distribuciones obtenidas, y la pérdida de la visión general del proceso.

De esta manera, se concluía que las nociones claras en ámbitos con un grado de abstracción bajo, no resultaban tan evidentes en la actual situación, y que a la hora de realizar futuras expansiones había que tener sumo cuidado con los detalles que se consideraran, si se quería que el proceso obtuviese una mejora apreciable.

En este punto, se procedió a la integración de las técnicas de estimación presentadas dentro del ámbito global del particionamiento (§6). Se explicó que si bien estos procesos de estimación habían resultado ser altamente eficientes, esta ventaja se podría perder debido a las intrínsecas deficiencias del particionamiento.

Esto era debido a que, aunque por supuesto los algoritmos estándar obtenían unos buenos resultados, encontraban dificultades ante ciertas características, como la presencia de altas comunicaciones entre las particiones hardware y software (§6.1).

De esta manera, se estudió el algoritmo de *Fiduccia-Mattheyses*, con el fin de que sirviera como muestra de los procesos de particionamiento, y se vio que una de sus virtudes era la alta exploración del espacio de diseño, gracias a la realización de movimientos simples en los que únicamente participaba un nodo simultáneamente.

Sin embargo, debido a estas características, el algoritmo encontraba serios problemas al tratar con comunicaciones altas. En este caso, el

movimiento de cualquiera de los nodos adyacentes a ellas supondría el rechazo de esta decisión, ya que la comunicación se activaría, incrementando mucho la carga temporal del sistema.

Por eso, se propuso un proceso de agrupamiento previo, en el que estas comunicaciones costosas se colapsaban, formando grupos que se movían como una entidad entre las distintas particiones (§6.3).

De esta manera, dichas comunicaciones permanecían siempre inactivas, y el algoritmo de particionamiento no se encontraba con los problemas mencionados. La única dificultad conceptual venía referida a cuál debería ser el criterio que permitiera formar estos grupos de una manera razonable.

Estaba claro que la selección de este criterio debía realizarse de una manera automática. En caso contrario, se estaría faltando al principio propuesto de minimización del tiempo de diseño. Sin embargo, al ser el criterio de agrupamiento un concepto complejo, dependiente de multitud de factores, se hacía muy difícil la labor de poder catalogarlo en forma de expresión matemática.

Así, características como la restricción temporal impuesta o la previsible situación de la solución requerida dentro del espacio de búsqueda inducían al criterio de agrupamiento a variar sin un patrón fijo predeterminado. Por lo tanto, y sobre la base de la experiencia práctica, se propuso una relación entre todos estos factores, que permitía predecir si una comunicación iba a resultar excesivamente costosa para el proceso de particionamiento (§6.3.1).

No obstante, se observó que la elección de un mal criterio constituía un hecho contraproducente, que podría conllevar la producción de un resultado peor que el que se obtendría en el caso del algoritmo estándar básico.

Para solventar esta desventaja, se propuso el perfeccionamiento de la aproximación mediante una técnica multi-etapa (§6.3.3), en la que, una vez

realizado el proceso de agrupamiento y particionamiento posterior, se liberaban las comunicaciones colapsadas en sucesivas iteraciones, permitiendo de esta manera que el sistema evolucionase de una forma más libre.

El caso extremo estaría conformado por el hecho de acabar liberando todas y cada una de estas comunicaciones, lo que produciría un grafo idéntico al manipulado en el caso inicial sin agrupamiento. En esta última situación se garantizaba que los resultados obtenidos nunca iban a ser peores que los previsibles en el caso clásico.

Por lo tanto, se conseguía mejorar el proceso de particionamiento, de tal manera que las técnicas de estimación integradas no perdiesen su eficiencia debido a problemas de otra índole.

En consecuencia, se ha logrado el propósito inicial de integrar el núcleo del proceso de Codiseño, dotándolo de unas características automáticas desde un punto de vista más abstracto al habitualmente utilizado en estos casos.

Sin embargo, y aunque el desarrollo matemático presentado se ha mostrado consistente, toda investigación elaborada en un marco tan práctico como el presente debe llevar implícito un nexo con la realidad que permita asociar los desarrollos a problemas concretos.

Por lo tanto, se realizó un exhaustivo proceso experimental (§7) en el que se ha tratado de poner de manifiesto todas las virtudes expuestas, tanto del módulo de estimación, como del de particionamiento. Para ello, se desarrolló un sistema de generación automática de diseños de prueba, con el que se ha permitido crear un gran número de ellos para su posterior tratamiento.

El hecho de disponer de una elevada muestra experimental es un hecho clave si se quiere que los resultados obtenidos tengan un carácter

general, y por lo tanto que sea posible extrapolar las conclusiones que se deriven de ellos.

Así, en la parte de las estimaciones, se llevaron a cabo una serie de pruebas en las que las tres técnicas presentadas, la plana, la escalar y la vectorial, debían obtener los parámetros de tiempo y coste asociados a los distintos grafos propuestos.

Estos resultados se compararon con los de la técnica básica sin reuso, a fin de calibrar el efecto de esta característica, y con los medidos por el *Behavioral Compiler*. Sobre todos estos valores, se llevaron a cabo estudios relacionados con la precisión, la fidelidad y el tiempo de proceso (§7.1).

Estas comparativas indujeron a la conclusión de que la técnica sin reuso es totalmente desaconsejable para ser integrada en un entorno real, debido a los malos resultados producidos en todos los ámbitos.

Las tres técnicas propuestas lograron reducir el error de precisión de una manera notable. Además, la técnica escalar se mostró más eficaz en este aspecto que la plana. Sin embargo, la vectorial, debido a los problemas comentados anteriormente de mala adecuación a la metodología macroscópica, no fue capaz de superar los resultados de la escalar.

En cuanto a la fidelidad, las tres obtuvieron unos resultados muy convenientes, y además altamente parecidos, por lo que las diferencias en este campo fueron mínimas.

Respecto al tiempo de ejecución, las tres técnicas redujeron en varios órdenes de magnitud el necesitado por el *Behavioral Compiler* para medir el valor real, lo cual tiene una gran importancia una vez integradas en el entorno de particionamiento. De ellas, la técnica plana resultó ser la más rápida, seguida de la escalar y de la vectorial.

El proceso experimental sobre el particionamiento fue encaminado hacia el estudio de la calidad ofrecida por la técnica de agrupamiento previa frente a la utilización directa del algoritmo clásico (§7.2).

En estas circunstancias se observó que mediante el agrupamiento, el sistema era capaz de alcanzar un número elevado de soluciones mejores, que en caso contrario no se hubiesen explorado. Es cierto que bajo ciertas circunstancias, se produjo la obtención de una pequeña parte de soluciones peores. Esto se solventó mediante la utilización de la técnica multi-etapa. Además, se comprobó que no era necesario llevar dicha técnica hasta su grado máximo de evolución, sino que situaciones intermedias podían proporcionar resultados interesantes.

Por otra parte, el tiempo de proceso necesario no se vio excesivamente influenciado por el uso de estas metodologías. Es más, en el caso de agrupamiento con una única iteración, este tiempo se logró reducir, debido a la disminución del número de entidades consideradas por el particionador.

En resumen, todas las metodologías presentadas han ofrecido unos buenos resultados, y más si se contempla la gran disminución que se ha producido en el tiempo de proceso. Por consiguiente, el objetivo básico de este trabajo, por el cual se pretendía la introducción de un nuevo marco de investigación, con rápidos métodos que solventasen la creciente complejidad de los problemas y con unos buenos niveles de calidad, ha sido cumplido.

8.2 Propuesta de trabajo futuro.

En esta última sección del capítulo, presentaré unas ideas acerca de posibles líneas de trabajo que extiendan la actual investigación. Por supuesto, las posibilidades son muchas, debido al amplio espectro de tareas que abarca el Codiseño. Sin embargo, detallaré aquellas que por su fácil adecuación a la metodología presentada, he creído más convenientes.

Por último, indicar que si bien las indicaciones que daré acerca de este posible trabajo futuro constituyen una forma que considero apropiada para

afrontar el problema, existen innumerables alternativas válidas que podrían ser utilizadas, dependiendo de los aspectos concretos sobre los que se pretenda realizar un mayor hincapié.

8.2.1. Consideración de nuevas características macroscópicas.

La metodología presentada, a raíz de la relación existente entre la formulación matemática y los resultados obtenidos, ha dado muestras de ser consistente. Sin embargo, esta cualidad de consistencia no es preciso que vaya unida, y de hecho no lo está, con la propiedad de completitud.

Así, se puede decir que las características consideradas, como son la posibilidad de elección de distintas implementaciones hardware, el paralelismo entre tareas o la capacidad de reuso, están bien modeladas, y se comportan de una manera acorde a como se esperaría en un caso real.

No obstante, es obvio que existen otros muchos detalles, relacionados con la naturaleza de los diseños, y que si bien no se han considerado para simplificar el desarrollo del presente trabajo, el marco presentado proporciona la posibilidad de incluirlos en futuras extensiones. De esta manera, es factible ir perfeccionando de una manera progresiva el actual entorno de Codiseño, para adaptarlo sucesivamente a nuevas características deseadas.

Así, un punto que se ha obviado es el tratamiento de los costes asociados a las interconexiones y multiplexores, elementos estos que sin duda tienen una repercusión importante en las características del diseño. De esta manera, los costes relacionados a estos elementos podrían extraerse del proceso de estimación de cada nodo, y añadirse a los referentes a las unidades funcionales, actualmente tenidos en cuenta. Entonces, podrían ser incorporados y tratados por las metodologías, con la inclusión de nuevas fórmulas que operasen sobre ellos (véase la sección siguiente).

Otro punto que sin duda se debería tratar es la consideración de los elementos secuenciales, tales como los registros o la unidad de control. Aunque en la presente aproximación han sido obviados, está claro que tienen una importancia en el coste final del sistema.

De esta manera, el número de registros de cada nodo por separado es fácilmente mensurable por la misma herramienta automática que proporciona el coste de las unidades funcionales. Entonces, sería necesario la extrapolación al conjunto global del grafo, realizando consideraciones acerca del previsible solapamiento entre las tareas.

En consecuencia, aquellos nodos que tengan lugar en etapas muy distintas dentro de la ejecución del sistema, podrán reusar los registros que contengan sus datos intermedios. Sin embargo, aquellos otros que se ejecuten en paralelo, deberán duplicar estos registros, aumentando el coste secuencial necesario.

Así, habría que estudiar la relación entre grado de paralelismo global del sistema, número de registros usados por los nodos individuales, y número final de registros requeridos por el diseño.

Por otra parte, sería también necesaria la consideración del coste introducido por la unidad de control. En este punto, habría que buscar una relación entre el número total de etapas del diseño, y la complejidad de la máquina de estados asociada que controle la ruta de datos.

Gracias a esto, el coste estimado sería mucho más ajustado al real, ya que no sólo se considerarían los elementos combinacionales, sino también los secuenciales.

Finalmente, cabe recordar que las extensiones permitidas no deben reducirse a estos aspectos, sino que es posible modelar cualquier característica real sobre la que se quiera incidir de una manera especial. Únicamente, recuérdese que todos estos procesos deberían llevarse a cabo

en el entorno macroscópico presentado, teniendo cuidado en no utilizar datos ni metodologías que pudieran disminuir la eficiencia de los procesos.

8.2.2. Estudio del efecto del reuso sobre el interconexionado.

Uno de los factores fundamentales que se han introducido en el presente trabajo es la consideración del posible reuso hardware entre los nodos que conforman esta partición. Sin duda, este factor da realismo a una aproximación, que en caso contrario adolecería de uno de los aspectos más destacados de la síntesis de sistemas reales.

Sin embargo, este mecanismo de reuso podría ser perfeccionado para que se adaptara mejor a otras características también presentes en la realidad. Una de las consideraciones que se podrían estudiar sería la influencia sobre el coste de interconexionado producido por la compartición de unidades.

Este reuso sin duda es beneficioso para la consecución de unos resultados en los que figuren el menor número de unidades funcionales posibles, lo que repercutirá en el bajo coste asociado al diseño final. No obstante, a medida que se va aumentando la compartición de las unidades, y por consiguiente produciéndose un ahorro de estas, también se está aumentando el número de posibles fuentes de datos que las están utilizando. En consecuencia, para organizar el mecanismo de acceso a estas unidades, es preciso disponer de un conjunto de multiplexores que se encarguen de dirigir los datos convenientemente.

A medida que el número de fuentes se ve incrementado, también la complejidad de esta multiplexación es aumentada. De esta manera, los sucesivos reusos de una cierta unidad producirán ahorros de costes cada vez más escasos, debidos a la compensación por parte de los multiplexores añadidos.

En esta situación, llegará un momento en el que un nuevo reuso introduciría efectos contraproducentes, ya que el ahorro de esa unidad será menor que el coste asociado a la incorporación de un nuevo multiplexor. Sobre todo, teniendo en cuenta que el aumento de coste producido por estos módulos no es lineal, llegados a un cierto tamaño.

Entonces, en este punto, es más conveniente introducir una nueva unidad funcional, y de manera explícita despejar y repartir la carga de conexionado que se había convertido en intratable.

Por lo tanto, sería conveniente realizar un estudio de la influencia del reuso sobre el coste final, caracterizando este hecho con un nuevo parámetro macroscópico. Así, se procedería a buscar la función que relacionase el coste de multiplexación respecto al número global de reusos producidos en el sistema.

De esta manera, se obtendría una representación más real de la conveniencia o no de compartir unidades funcionales.

8.2.3. Caracterización más precisa de los nodos: la disgregación.

En los puntos anteriores se ha tratado acerca de cómo mejorar las características que reflejan las situaciones reales en los diseños, con la introducción de nuevos parámetros capaces de modelar distintos aspectos físicos.

Sin embargo, se pueden realizar correcciones mucho más sutiles, que influyan, no ya en los elementos asociados a cada nodo, sino en la estructura intrínseca de los mismos.

Así, hasta ahora se ha considerado que los nodos son elementos monolíticos, que contienen una serie de operaciones, y que se estiman *a priori* en su totalidad, con el fin de que la información obtenida sirva de base

para la caracterización del sistema completo, en lo que se ha denominado la estimación de grafo.

No obstante, una vez que se ha realizado el proceso de particionamiento, decidiéndose por completo la composición de ambos subsistemas, la descripción correspondiente al hardware es introducida en una herramienta de Síntesis de Alto Nivel, con el fin de generar de una forma automática el circuito asociado a esta partición.

De esta manera, lo que *a priori* se estaba considerando como una colección de nodos, ahora se corresponde con una descripción plana, en la que ya no existe ningún tipo de distinción entre las diferentes tareas que la conforman. Así, los nodos constituirían entidades ficticias, que tienen utilidad como mecanismo de jerarquía durante el proceso de Codiseño, pero sin ningún significado real a la hora de sintetizar el diseño final.

Por lo tanto, la herramienta de Síntesis de Alto Nivel podría tomar decisiones en las que las operaciones hasta ahora contenidas en un mismo nodo, pasen a planificarse en etapas muy dispersas dentro del periodo de ejecución total del circuito.

Estas decisiones pueden estar plenamente justificadas, debido a que de esta forma se evite el solapamiento de unidades funcionales costosas, redundando en un menor coste final del circuito.

Supongamos, por ejemplo, que se tienen dos nodos con una dependencia de datos entre ellos. En el caso general, el segundo no podrá ejecutarse hasta que no acabe el primero, forzando a un tratamiento secuencial de ambos.

Sin embargo, puede darse el caso de que las operaciones del primer nodo produzcan datos que necesitan ser consumidos por *parte* de las operaciones del segundo nodo. En cambio, puede ocurrir que en este existan *otras* operaciones que no se vean influidas por estas dependencias.

En este caso, el primer grupo de operaciones sí debe esperar a la finalización del nodo antecesor, pero el segundo grupo puede ejecutarse libremente antes de que esto suceda. Por lo tanto, aquí sí estará permitido cierto grado de paralelismo, que en el caso general no se hubiese estimado.

De esta manera, se produce el efecto comentado al principio, en el que las distintas operaciones dentro de un mismo nodo pueden ser finalmente planificadas en ciclos de reloj muy distantes, debido a las particulares relaciones entre ellas.

En consecuencia, sería preciso estudiar, siempre desde el punto de vista macroscópico, cómo estaría compuesto cada uno de los nodos, y qué facilidad tendrían las operaciones internas de desligarse unas de otras, aumentando su rango de movilidad. Este fenómeno podría denominarse *disgregación*, y constituiría una nueva fuente de información para aplicar a los procesos de estimación asociados.

Así, se requeriría un estudio de las distintas unidades funcionales contenidas en cada nodo, y de las dependencias entre ellas y entre los parámetros comunicados desde el exterior. Si estas unidades están muy relacionadas, y casi todas necesitan la transmisión completa de los datos por parte de los nodos antecesores, entonces el índice de disgregación sería bajo. Si, por el contrario, existen bloques de operaciones claramente separados, sin una interacción determinada, y que en muchos casos puedan ejecutarse antes de la recepción de datos, el índice de disgregación sería alto.

El siguiente problema consistiría en determinar cómo afecta este parámetro a los modelos de estimación presentados. De esta manera, las posibilidades de reuso de un nodo con alta disgregación se deberían incrementar de una manera proporcional, ya que las unidades internas pueden tender a dispersarse más para evitar posibles conflictos, y por lo tanto, aumentando la capacidad de ser compartidas.

Un estudio interesante sería determinar la relación clara de este parámetro con el resto del modelo, indicando las expresiones detalladas que lo harían integrarse en el conjunto de métodos macroscópicos. Esto llevaría a una disminución del error en las estimaciones, debido a que se considerarían aspectos reales del proceso de síntesis final del circuito.

8.2.4. Estudio de la planificación de las comunicaciones en el bus.

Un problema que se puede presentar, y que no se ha considerado en el presente trabajo, es el asociado a las características finitas del bus del sistema. Evidentemente, este es un elemento esencial dentro de la arquitectura objetivo, ya que es el que ha de soportar la carga de comunicaciones entre las dos particiones, la cual puede llegar a ser muy elevada en ciertos momentos.

De esta manera, el bus físico tendrá una ancho de banda determinado, y será imposible transmitir información a una mayor frecuencia, lo que puede influir en la disponibilidad de los datos en ambas particiones.

En el trabajo presentado, se ha introducido el concepto de ejecución paralela entre el componente hardware y el software. Esto siempre se ha considerado así, ya que este hecho no supone ninguna desventaja respecto a las posibilidades de reuso de unidades funcionales, al contrario de lo que ocurre con el paralelismo entre las distintas tareas hardware, el cual sí tiene influencia.

Así, el paralelismo hardware-software sólo se ha limitado por la posibilidad de dependencias de datos entre ambas particiones, equivalentes a las comunicaciones de parámetros a través del bus. En la estructura interna, este tipo de dependencias se agrupaban en el conjunto de aristas d) (§4.3.2), que eran las que cruzaban la frontera imaginaria entre las dos particiones.

El hecho de atravesar esa frontera, lleva implícito el uso del bus, que es un recurso limitado. Sin embargo, no se ha considerado el hecho de que dos comunicaciones, que se han de llevar a cabo simultáneamente[1], necesiten una capacidad de transmisión superior a la proporcionada. De esta manera, el tiempo de transmisión asociado a estas comunicaciones, se verá incrementado por un tiempo de espera, por lo que la tarea que requiere los datos sufrirá un retraso no considerado previamente.

Por lo tanto, se precisa de un proceso de planificación, impuesto por la naturaleza finita del bus, y que tenga como finalidad la ordenación racional de la realización de estas comunicaciones, de tal forma que repercuta lo mínimo posible en el tiempo de ejecución global del sistema.

Según los distintos casos, existirán comunicaciones que no haya más remedio que realizarlas antes que otras. Esto se producirá cuando entre los nodos adyacentes a ellas exista una ordenación natural. De esta manera, si un nodo hardware requiere datos de un nodo software, y a su vez, proporciona resultados a otro nodo software distinto, la primera comunicación se ha de realizar forzosamente antes que la segunda.

Sin embargo, habrá ocasiones en que las distintas comunicaciones puedan ejecutarse en paralelo, y por lo tanto, será preciso estudiar cuál de ellas debería realizarse en primer lugar. Así, lo ideal sería desarrollar un algoritmo de optimización, que dadas las distintas permutaciones entre las comunicaciones sin dependencias, determine su ordenación, de tal forma que el sistema resultante tenga la mejor relación de tiempo y coste. Para ello, se podrían estudiar criterios similares a los tratados en la planificación macroscópica de los nodos.

[1] Por ejemplo, en el caso de que un nodo software haya de proporcionar datos a dos tareas hardware que se ejecutan en paralelo. Las dos comunicaciones compiten por el uso del bus, ya que ambas son prioritarias.

Además de esto, también podría ser interesante la ampliación del concepto clásico de comunicación. Así, tradicionalmente se ha venido reservando este nombre para la transmisión de parámetros entre el hardware y el software. Sin embargo, las propias tareas software también necesitan acceder a memoria para leer sus datos, y por lo tanto, deben hacer uso del bus.

El hecho de considerar un número suficiente de registros en el procesador para almacenar esta información, puede resultar demasiado simplista, especialmente en problemas de alta complejidad. En consecuencia, se podría estudiar el impacto sobre el bus debido a esta necesidad de realizar sucesivas lecturas de datos por parte del subsistema software.

Igualmente, se podría llevar a cabo un estudio acerca del ancho de bus más conveniente. Este parámetro, que incidiría en el tiempo asociado a las comunicaciones, podría conseguir la obtención de diseños más rápidos, a costa de introducir buses más costosos. El estudio de este equilibrio entre los dos factores también podría conducir a conclusiones interesantes.

8.2.5. Modelado del componente software.

Como se ha explicitado a lo largo del presente trabajo, se ha obviado la utilización de unos procesos de estimación relacionados con la partición software, de tal manera que pudiesen proporcionar unos valores característicos adecuados de las tareas que se ejecutan en este subsistema.

Por lo tanto, una posible línea de investigación sería el estudio de esta partición, y la propuesta de métodos que pudiesen integrarse en el sistema presentado. Por supuesto, estos métodos deberían verse favorecidos por la introducción de los conceptos macroscópicos. Evidentemente, hoy en día existen muchas metodologías de estimación software, pero que también

están relacionadas con conceptos microscópicos, de igual manera que ocurría con las estimaciones hardware.

Asimismo, las propuestas de trabajo basadas en arquitecturas más complejas deberían tenerse muy en cuenta, de tal manera que pudiesen ir introduciéndose en los sistemas más típicos de Codiseño, con el fin de poder abordar problemas muy restrictivos a corto plazo.

En consecuencia, se debería trabajar en la integración de sistemas con memoria cache, que permitiesen un más rápido acceso a los datos, y por lo tanto, en el desarrollo de estimadores capaces de afrontar esta innovación. Así, la predicción de cadenas de referencias a memoria, y los porcentajes de acierto en los accesos a la cache, tendrían que ser objeto de estudio prioritario.

Además, el uso de procesadores con ejecución segmentada supondría una gran mejora en el rendimiento de los sistemas producidos, lo que conllevaría el tratamiento de la predicción de los datos necesarios, así como del flujo de control.

Estas líneas de investigación parecen un poco alejadas del presente trabajo, el cual se centra de manera casi exclusiva en los detalles concernientes a la partición hardware. Sin embargo, estas decisiones tendrían una gran repercusión en este subsistema, por lo que deberían ser consideradas posteriormente.

Así, la evolución de la complejidad de ambas particiones tendría que llevarse a cabo de una manera simultánea, que permitiese la buena sincronización entre ambas, y que al final, diese como resultado un sistema perfectamente integrado.

Aparte de estas, existen muchas otras posibilidades de extender el trabajo aquí presentado. Esto es debido a la gran versatilidad del campo de Codiseño, que tendría que ser explotada progresivamente, con el fin de abordar problemas de síntesis de sistemas cada vez más complejos.

El generador aleatorio de grafos.

Apéndice A

El primer apéndice de este trabajo está dedicado a la explicación del generador aleatorio de diseños de prueba, utilizado en el capítulo dedicado a las pruebas experimentales. Es fundamental que se conozcan los mecanismos que han servido de base a la realización de estos experimentos, no sólo con afán de ofrecer mayor claridad, sino para favorecer la reproducción de los mismos en iguales circunstancias.

Este punto resulta clave en la investigación, ya que la disponibilidad de bancos de pruebas comunes facilita la comparación de resultados, y acerca puntos de interés afines a toda la comunidad científica inmersa en un mismo proyecto.

A.1 Características del generador.

El proceso de generación de grafos tiene asociadas dos particularidades. Por una lado, es una tarea totalmente automática. Gracias a esto es posible la creación muy rápida de un gran número de diseños de prueba, lo cual favorece la capacidad de generalización de los resultados obtenidos.

Por otra parte, dispone de un mecanismo de ejecución parametrizable. Así, se puede elegir de una manera bastante precisa la forma y características del diseño producido.

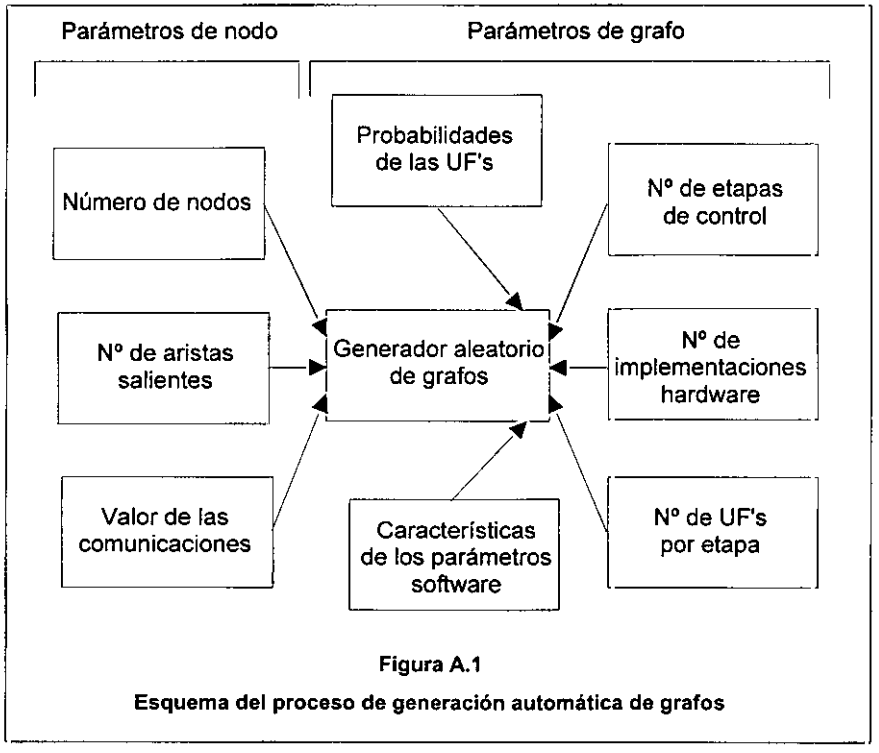


Figura A.1

Esquema del proceso de generación automática de grafos

Se pueden distinguir dos conjuntos de parámetros especificables por el usuario, los denominados *parámetros de grafo* y los *parámetros de nodo*, gracias a los cuales se podrá guiar el proceso de la forma más conveniente.

Los primeros hacen referencia a la estructura externa del grafo, como el mapa de conexión entre los distintos nodos y las características de las comunicaciones que los enlazan. Los segundos están relacionados con la estructura interna de cada nodo y sus diversas implementaciones hardware, especificando propiedades esenciales como los porcentajes de distribución de las distintas unidades funcionales.

El esquema general del proceso se puede observar en la Figura A.1. En la próxima sección se ofrece una reseña detallada de cada uno de estos parámetros.

A.1.1. Parámetros de grafo.

a) *Número de nodos.* Como su nombre indica, este parámetro fuerza al diseño generado a disponer de tantos nodos como marque su valor. Es quizás el dato más importante a la hora de clasificar el ejemplo obtenido, ya que delimita el tamaño, y por tanto la complejidad, del grafo que lo representará.

b) *Máximo y mínimo número de aristas salientes por nodo.* Indican la conectividad y estructura final del grafo. De esta manera, cada uno de los nodos dispondrá de un número aleatorio de sucesores, generado entre los límites marcados por estos dos parámetros.

c) *Máximo y mínimo valor de las comunicaciones y desviación.* Una vez generada la estructura de dependencias entre los nodos, la cual delimita la jerarquía esencial del grafo resultante, es preciso asignar un peso de comunicación a cada arista, de tal forma que indique la cantidad de datos que los nodos adyacentes intercambiarán, en caso de asignarse a particiones distintas.

Esto representará el tiempo de transmisión durante el cual se hace uso del bus del sistema, lo que puede ocasionar una gran repercusión sobre el proceso final de particionamiento. De esta manera, el peso asociado se genera aleatoriamente entre los dos límites máximo y mínimo introducidos. Un valor del máximo muy elevado indicaría una gran interacción entre las tareas, y por lo tanto, un flujo de información muy alto a través del bus.

Asimismo, además de estos dos límites, se cuenta con el parámetro denominado *desviación*. Este parámetro viene determinado por un número real entre cero y uno, e indica la dispersión generada aleatoriamente respecto al centro del intervalo requerido, delimitado por los valores de máximo y mínimo.

Así, un valor de cero en la desviación hace que se generen valores de comunicaciones homogéneos entre el máximo y el mínimo introducidos de tal manera que todos los puntos intermedios tienen igual probabilidad de ser elegidos.

Por el contrario, a medida que este factor tiende a uno, los valores de comunicación próximos al máximo y al mínimo van aumentando sus probabilidades de ser generados, mientras que los valores intermedios las van disminuyendo.

De esta forma se pueden obtener pesos de comunicaciones con valores altamente separados, lo que resulta muy útil a la hora de estudiar su efecto en los algoritmos de particionamiento.

A.1.2. Parámetros de nodo.

a) *Máximo y mínimo número de etapas de control.* Estos parámetros indican los límites del tamaño de cada uno de los nodos generados por el sistema. De esta manera, es posible controlar los rangos de tiempos de ejecución que se verán presentes en el diseño de prueba, lo cual está asociado al nivel de complejidad de las tareas relacionadas.

Así, un límite máximo muy alto indicará la existencia de módulos que requieren un elevado tiempo de proceso, y que por lo tanto extienden su ejecución durante un largo periodo. Esto va unido a la mayor probabilidad de solapamiento entre los distintos nodos del grafo, y de igual manera, a una más alta influencia de las técnicas de reuso presentadas.

Si los límites máximo y mínimo vienen dados por valores muy separados, el efecto producido es el de coexistencia de tareas muy complejas con otras más sencillas, con lo que es posible dar un aspecto de heterogeneidad al diseño creado.

Estos valores extremos están siempre asociados a la implementación básica del nodo, aquella que es generada siempre por defecto. En el caso

de que se deseara más de una implementación por nodo, como se verá más adelante, no se respetarían estos dos límites, ya que para las sucesivas implementaciones sería necesario aumentar el número de etapas a fin de reducir los distintos niveles de paralelismo interno asociados a las unidades funcionales.

b) *Máximo y mínimo número de unidades funcionales por etapa.* Estos valores delimitan la ejecución paralela de unidades funcionales dentro de cada una de las etapas de control disponibles.

Así, un valor alto para el máximo indicaría la posibilidad de generar un gran número de unidades con una ejecución simultánea, con lo que el coste final del nodo se elevaría en gran manera. Por el contrario, si este valor es bajo, el grado de paralelismo disponible en cada momento no sería muy alto, con lo que las posibilidades de reuso interno se verían acrecentadas, conllevando una reducción del coste.

Es precisamente este factor el que más se ve influenciado por la elección de estos parámetros. Si, por ejemplo, se decide generar un nodo con un alto número de etapas de control, pero por el contrario el número de unidades por etapa se fuerza a mantenerse bajo, el coste del nodo presumiblemente no será muy elevado, debido al efecto del reuso. Sin embargo, si se obliga a que crezca el número de unidades, la repercusión en el coste final será muy notable.

Lo mismo que ocurría en el apartado anterior, estos parámetros están relacionados con la implementación básica de cada nodo. Para la generación de sucesivas implementaciones, los grados de paralelismo se irán eliminando, por lo que podrían salirse de los márgenes fijados.

c) *Probabilidades de las distintas unidades funcionales.* Este es uno de los factores fundamentales que afectan a la formación de los nodos, ya que tiene relación directa con su distribución interna.

Previamente a la generación automática del grafo de diseño, es necesario la elaboración de una biblioteca de unidades funcionales que sirva al proceso como base de datos acerca de los distintos módulos disponibles para la creación de los nodos.

Esta biblioteca ha de disponer de una lista de las unidades funcionales que se pueden seleccionar, junto con sus características intrínsecas, como son el tiempo de ejecución, el coste y la operación que pueden realizar.

La elección de la biblioteca depende de las circunstancias que se quieran poner de manifiesto con la creación del diseño pertinente. Así, si lo que se desea es estudiar los efectos del reuso en un sistema, la biblioteca ha de disponer de muy pocas unidades distintas, para que la composición de los nodos sea muy homogénea, facilitando de esta manera su compartición. Por el contrario, si se quiere dotar al proceso de mayor realismo, es posible añadir cualquier número de módulos que se considere oportuno.

Lo mismo ocurre con los parámetros de tiempo y coste, que pueden ser ajustados libremente para conferir a los nodos unas características apropiadas para la realización de los estudios elegidos.

La presencia de esta biblioteca es algo externo al propio sistema de generación automática, lo cual permite la coexistencia de varias de estas entidades, con características muy distintas, y usar en cada momento la opción que se crea más oportuna. Así, es posible disponer de bibliotecas ya existentes extraídas de herramientas comerciales, lo que conllevaría la generación de nodos muy reales, o bien se podría optar por crear una simplificada, con características ideales.

Una vez escogida la biblioteca de la que se van a extraer las unidades funcionales contenidas en ella, es preciso indicar cuál es la probabilidad de distribución de cada una de ellas dentro del sistema generado.

De esta manera, mediante una lista de porcentajes asociados a cada uno de los tipos de módulos, se va forzando a que los distintos nodos tengan una composición determinada. Utilizando este mecanismo, es posible modelar tareas de las que se conoce su comportamiento general, pero de las que se ignora su composición detallada.

d) *Número de implementaciones hardware.* Como se ha venido comentando a lo largo del presente trabajo, una de las características incluidas en la aproximación es la posibilidad de trabajar con múltiples implementaciones hardware asociadas a cada nodo.

De esta manera, mediante el uso de este parámetro, es posible elegir exactamente dicho factor, lo que llevaría a generar distintas distribuciones de unidades funcionales para cada uno de los nodos.

Si el número de implementaciones que se elige en un determinado caso es de uno, entonces el generador realizará una asignación de unidades funcionales, según las probabilidades asociadas a cada uno de los tipos (véase el apartado anterior), creando una implementación óptima en tiempo.

Sin embargo, si el número elegido es de dos, entonces el sistema toma el conjunto de unidades asignadas al nodo, y elimina todos los niveles de paralelismo interno existentes, mediante la incorporación de nuevas etapas de control donde sea necesario.

En consecuencia, se obtendrá una segunda implementación en la que el coste asociado se ha disminuido al mínimo, aumentando para ello el tiempo de ejecución del nodo.

Para mayor número de implementaciones, se calculan situaciones intermedias a los casos más rápido y más barato. Para ello, se van eliminando distintos niveles de paralelismo de una forma progresiva.

El resultado final que se obtiene es un conjunto de implementaciones distintas asociadas a cada nodo, las cuales se suponen equidistribuidas a lo largo del espacio de posibilidades.

e) *Características de los parámetros software.* Estos valores representan las propiedades que tendrán las implementaciones software asociadas a los distintos nodos, como representación idealizada de esta partición.

Como ya se ha venido comentando, no se ha trabajado con estimadores relacionados con este subsistema por motivos de simplificación, aunque es necesario disponer de los parámetros asociados para poder llevar a cabo el proceso de particionamiento.

Esto se consigue mediante una generación ponderada de dichos parámetros, respecto a las características globales relacionadas con la partición hardware, inherentes a cada uno de los nodos.

De esta manera, tareas que requieren un tiempo elevado para la ejecución de sus funcionalidades en hardware, tendrán también asociado un tiempo alto relacionado con su implementación software, y lo mismo ocurre en el caso contrario.

Así, se especifica el *factor de orden* como el parámetro que indica cuán grande debe ser el tiempo software comparado con la media de los tiempos asociados a las distintas implementaciones hardware. Se toma la media para realizar este proceso debido a la gran variedad de posibles implementaciones hardware, las cuales, dependiendo de la disposición interna de las unidades, tendrán asociados distintos valores de sus tiempos de ejecución. De esta forma, se consigue modelar este parámetro según las necesidades propias de cada diseño.

Asimismo, se ha incorporado un nivel de aleatoriedad, denominado *factor de desviación software*, que tiene la función de matizar el valor del tiempo obtenido mediante la especificación del factor de orden anterior.

Este nuevo parámetro viene indicado mediante un porcentaje, que determina el margen superior e inferior respecto del valor previamente calculado, en el que se generará el tiempo final software.

En consecuencia, es posible dar un efecto variable a este proceso, y escapar de la excesiva linealidad de obtener un tiempo software como un múltiplo entero de la media de los tiempos hardware.

Respecto al coste software, y como ya se indicó previamente, se considera que es nulo, por lo que no requiere ningún tipo de matización ulterior.

El *Behavioral Compiler*: Ventajas e inconvenientes.

Apéndice B

En el apartado experimental del presente trabajo se ha mencionado el uso de la herramienta automática de diseño denominada *Behavioral Compiler* [Knap96]. Esta utilidad es uno de los desarrollos básicos integrados dentro del entorno comercial de *Synopsys*. Se ha optado por el uso de la mencionada utilidad con el fin de proceder a la obtención de un conjunto de medidas reales en un proceso lo más estándar posible.

De esta manera, se crea un punto de referencia para futuros trabajos, en los que las tareas de comparación de resultados se simplifican enormemente, debido a la gran presencia, creciente día a día, de esta herramienta. Así, de igual forma que en esta investigación se ha tratado de ofrecer una serie de metodologías que mejoren el tiempo de proceso del *Behavioral Compiler*, manteniendo la calidad relativa lo más alta posible, las futuras propuestas podrán ser fácilmente cotejadas con la presente, mediante la comparación de sus resultados con los de esta herramienta.

En este apéndice se pretende dar una idea general de cómo se han aprovechado las ventajas proporcionadas por el *Behavioral Compiler*, y de qué manera se han solventado las carencias que presenta.

B.1 Características del *Behavioral Compiler.*

Como su propio nombre indica, esta herramienta proporciona el entorno adecuado para la síntesis de sistemas especificados mediante una descripción de comportamiento. Esta característica ofrece una ventaja muy importante respecto de la tradicional descripción estructural, en la que se debía especificar la presencia e interconexión de todos los componentes existentes en el diseño.

De esta manera, y empezando con un código escrito en VHDL (aunque la opción de empleo de *Verilog* también se contempla), es posible aplicar distintas restricciones relativas al tiempo y área final de diseño, que guían al proceso hacia la mejor solución posible, de acuerdo a las necesidades de cada caso.

El resultado final es una información detallada de todo el trabajo realizado. Así, se obtienen los datos correspondientes a los ciclos de reloj necesarios para ejecutar el diseño, y al área combinacional y secuencial empleada en su generación.

De igual forma, se puede acceder a una lista de todas las unidades funcionales empleadas, así como de la planificación de operaciones en los distintos ciclos de reloj. Otro tipo de opciones contemplan la posibilidad de realizar estudios acerca de la vida de las variables utilizadas, y por lo tanto del aprovechamiento de los registros, y de características detalladas relativas a la máquina de estados asociada al controlador.

Toda esta información posibilita el conocimiento exacto de cada uno de los diseños que se pretenden estudiar, y un subconjunto de ella es la que ha sido usada en §7 al presentar los resultados experimentales medidos para el valor verdadero.

Por supuesto, una característica más que ofrece el *Behavioral Compiler* es la conexión con otras herramientas comerciales que operan a un nivel de abstracción mucho menor. De esta forma, existe la posibilidad de

seguir trabajando con el sistema y llegar a conseguir diseños a nivel físico, con lo que se abren posibilidades en el campo del prototipado, pudiendo extender el proceso de Codiseño más allá del límite tratado en el presente trabajo.

B.2 Carencias del *Behavioral Compiler.*

Evidentemente, y aunque el *Behavioral Compiler* proporcione un buen entorno para la verificación del trabajo experimental propuesto, posee ciertas carencias que es preciso solventar.

Estas deficiencias no se deben tanto a la propia naturaleza de la herramienta, sino al uso que se pretende hacer de ella, que no equivale en su totalidad a los propósitos para los que fue creada.

Uno de los puntos principales propuestos a lo largo de este trabajo, es el impacto que tiene el reuso hardware dentro del cálculo del coste global asociado a un diseño. De esta manera, las pruebas ideadas han sido encaminadas en la dirección de corroborar este hecho, para darle así una consistencia experimental.

En consecuencia, es obvio que las mediciones reales efectuadas para contrastar los valores estimados consideren intrínsecamente esta *capacidad, para que los resultados obtenidos posean la misma base de* suposiciones, y por lo tanto sean comparables.

La jerarquización propia del *Behavioral Compiler*, heredada de su dependencia del VHDL, está organizada alrededor del *proceso*, como unidad mínima de tratamiento. De esta manera, las distintas unidades contendrían aspectos relevantes extraídos de la especificación inicial, y se sincronizarían mediante el uso de parámetros, lo que daría al diseño un aspecto modular.

Esta característica es muy útil a la hora de explicitar un circuito, ya que dota al proceso de una facilidad, en cuanto a la depuración y estudio del

mismo, altamente deseable. Dicha situación es análoga a la presentada en las descripciones software que hacen uso de una jerarquía de procedimientos, mediante la utilización de la programación modular.

Por lo tanto, lo más lógico, a la vista de lo comentado, sería diseñar cada uno de los nodos del grafo que se está tratando de medir como un proceso independiente, de tal manera que tuviera una cierta entidad propia.

Sin embargo, esta consideración acarrea un problema no deseado respecto a los propósitos relacionados con la presente investigación. El *Behavioral Compiler* no considera la posibilidad de reuso de unidades funcionales entre los distintos procesos que conforman una entidad. En consecuencia, optar por esta estructura de diseño sería privar a los resultados del beneficio de este factor, lo que los haría incompatibles con las estimaciones relacionadas producidas por las aproximaciones presentadas.

Este hecho de no considerar el reuso no debe asociarse a una deficiencia intrínseca de la herramienta. Es normal que esto ocurra, por la propia naturaleza de los procesos en VHDL. De esta forma, dichos procesos tendrían un comportamiento asíncrono entre sí, ejecutándose libremente cada vez que ocurriera un determinado evento que los disparase. Por lo tanto, no están sujetos a una planificación inicial, sino que el instante de comienzo se producirá por condiciones estocásticas no sujetas a conocimiento previo.

Si se tiene en cuenta esta propiedad, es fácilmente asumible que no se considere el reuso entre los distintos procesos, ya que de ninguna manera sería estimable de una forma eficiente. Esto se debe a que la posibilidad de compartir unidades es totalmente dependiente del solapamiento existente entre los distintos módulos. Como este solapamiento es función de la planificación final, y esta no se conoce en una primera instancia, el reuso entre procesos es algo no predecible.

En consecuencia, y partiendo de la opción ya tomada de utilizar esta herramienta, es preciso adaptarla de tal manera que permita la medición de características de los diseños de prueba, a la vez que se hace uso de la posibilidad de compartir hardware entre las distintas tareas.

Así, parece claro que la única opción disponible es la inclusión de todo el código asociado al diseño dentro de un mismo proceso. Esto conlleva varios problemas, que hacen más difícil la comprobación de los resultados experimentales, pero es la única solución, considerando los elementos de los que se dispone.

Un primer contratiempo es la complejidad y tamaño del código obtenido. Al forzar la coexistencia de la totalidad de las funcionalidades en un solo proceso, se acaba generando un programa VHDL difícil de tratar y que pierde toda la jerarquía y estructuración que inicialmente poseía. Esto, aunque representa una dificultad más práctica que teórica, se opone al concepto deseado de separación modular.

Por otra parte, existe el problema de la limitación de las capacidades de la herramienta utilizada. De esta manera, el *Behavioral Compiler* posee un máximo número de operaciones que puede manejar dentro de un mismo proceso, creando un umbral más allá del cual no es posible aumentar la complejidad del diseño tratado.

El límite observado a raíz del proceso experimental sitúa entre 500 y 600 este número máximo de operaciones, para el entorno utilizado basado en una máquina Ultra-Sparc, y dependiendo de circunstancias particulares del problema. Así, si el diseño introducido tiene pocos grados de libertad, o en otras palabras, los rangos de movilidad de las distintas operaciones son reducidos, entonces la herramienta no requiere de mucha memoria para sus cálculos, y el número de operaciones puede ser mayor. Sin embargo, si es preciso explorar un elevado número de posibilidades de planificación y asignación, el total de operaciones aceptadas disminuye drásticamente.

Con todas estas limitaciones, se ha optado por la agrupación del conjunto de nodos dentro de un mismo *proceso*, a fin de explotar todas las ventajas existentes en cuanto a las posibilidades de reuso hardware. Por supuesto, esta tarea se ha automatizado de tal forma que consuma el mínimo tiempo posible, y en consecuencia no afecte de una manera decisiva al tiempo total de evaluación de los experimentos.

Para ello, se ha desarrollado un programa mezclador, que partiendo de las descripciones VHDL de los distintos nodos[1], los va combinando dos a dos en un mismo archivo de proceso.

Esta operación se realiza de tal manera que se respeten los detalles de planificación macroscópica originados por las técnicas de estimación desarrolladas. Así, si un nodo debe comenzar su ejecución un cierto número de unidades de tiempo después que otro, es precisamente esa diferencia la que se utiliza como desplazamiento en el programa mezclador, y los códigos VHDL asociados a los nodos finalmente aparecerán distanciados ese mismo número de etapas.

Un punto importante es que aunque se fuerza a que las descripciones de los nodos posean este desplazamiento relativo, esto sólo constituye una circunstancia coyuntural. Posteriormente, el *Behavioral Compiler* tendrá total libertad para mover las operaciones internas y optar por la planificación final que considere más oportuna.

En consecuencia, y a modo de resumen, para solventar la deficiencia inherente de la herramienta, y partiendo de las descripciones de los nodos individuales, se realiza una etapa de mezcla en la que se obtiene un único proceso VHDL, donde los nodos anteriores aparecen desplazados distintos

[1] Que previamente también han sido creadas de una forma automática a partir de la estructura interna proporcionada por el generador aleatorio de grafos.

ciclos de reloj, dependiendo de sus tiempos de comienzo decididos previamente por el proceso de planificación.

B.3 Utilización de los recursos del *Behavioral Compiler.*

Tras la obtención del proceso que contiene toda la información necesaria asociada al diseño de prueba, es necesario indicar a la herramienta las particularidades de la labor de medición que se desea realizar.

Un proceso necesario consiste en la especificación clara de las unidades funcionales que se quieren utilizar, como reflejo fiel de la biblioteca de módulos que se considerará en las técnicas de estimación y particionamiento posteriores. Esto se consigue ligando cada una de las operaciones que aparecen en la descripción del diseño, a los posibles operadores considerados con capacidad para implementarlas. La realización de esta tarea se lleva a cabo mediante el uso de la directiva denominada *map_to_operator*, estándar en el entorno de diseño del *Behavioral Compiler.*

Posteriormente, se llevan a cabo las operaciones típicas de análisis, elaboración y selección del ciclo de reloj,

```
analyze -f vhdl archivo
elaborate -s entidad
create_clock clk -p periodo
```

y más tarde se procede a forzar al sistema para que no practique ningún tipo de encadenamiento de unidades funcionales, mediante la expresión:

```
dont_chain_operations {operaciones}
```

Aquí, el campo *operaciones* delimitaría todas las unidades del diseño. El motivo de no permitir el encadenamiento reside en mantener el proceso

experimental dentro de unos límites de simplicidad. En aproximaciones sucesivas se podría tener en cuenta esta característica.

Después, es preciso forzar a que se cumplan las restricciones del propio diseño. Como la herramienta tiene libertad para mover las distintas operaciones en busca de la solución más ventajosa, hay que indicar de alguna manera que respete aquellas contenidas dentro del camino crítico.

Esto es debido a que, aunque no sea objeto de trabajo de la herramienta, existe una partición software que crea una serie de dependencias con el circuito hardware, y por lo tanto se requiere una temporización entre los dos subsistemas que haga fluir los datos de una forma razonable. Esas dependencias pueden forzar a que determinadas operaciones hardware se ejecuten en ciertos ciclos de reloj, y no en otros, a fin de que el proceso global no se retrase (o dicho de otra manera, es necesario que se respete el camino crítico obtenido en la planificación macroscópica).

Así, antes de realizar ninguna tarea ulterior con la herramienta, hay que indicar qué operaciones se deben realizar en qué ciclos de reloj, mediante una secuencia de órdenes del tipo:

```
preschedule {operaciones} ciclo
```

De esta manera, se tiene la certeza de que las restricciones serán cumplidas. Después, es preciso indicar al *Behavioral Compiler* que busque la planificación óptima asociada al diseño y a las condiciones impuestas previamente.

Esta herramienta, dispone de varios modos de planificación, en los que se le permiten distintos grados de libertad a la hora de recorrer el espacio de soluciones. En este caso, se ha considerado oportuno elegir el modo de mayor libertad, que si bien consume un tiempo de proceso más elevado, tiene más posibilidades de encontrar una planificación óptima. De esta manera, se realiza:

```
schedule -io free_floating -area_based
```

La última opción indica a la herramienta que lo que se pretende es maximizar el reuso de unidades hardware, y por lo tanto optimizar el coste, siempre sin exceder del número de ciclos de la planificación.

Tras esta operación, se obtienen los parámetros medidos relativos al diseño, como el tiempo, coste o distribución de unidades funcionales, que servirán como valores reales en el proceso de comparación con los proporcionados por las técnicas de estimación propuestas.

Por último, indicar que toda esta secuencia de operaciones *desarrollada sobre el Behavioral Compiler se ha realizado mediante un* proceso automático, integrado en el resto del conjunto de pruebas experimentales.

Estudio de un ejemplo real: El codificador de vídeo H.261

Apéndice C

Este último apéndice está dedicado a la aplicación de las técnicas de estimación y particionamiento desarrolladas a lo largo del presente trabajo sobre un ejemplo extraído a partir de una aplicación real. El motivo fundamental de este estudio es comprobar cómo se comporta la aproximación propuesta al ser aplicada en casos de una complejidad alta, considerando para ello el número de operaciones internas de las que están compuestos.

Además es preciso verificar que los resultados obtenidos en el capítulo de trabajo experimental, y que fueron extrapolados elevándolos al nivel de generalidad, se siguen manteniendo ahora, cuando en lugar de un amplio rango de pruebas se dispone únicamente de un ejemplo aislado

C.1 Características del codificador H.261

Una primera consideración que debe hacerse antes de la exposición detallada del ejemplo utilizado, es que ha sido creado a partir de un caso real presente en las aplicaciones prácticas. Dicho de otra manera, el ejemplo no es real, sino que está basado en un proceso que sí lo es.

Con esto se quiere dejar claro que el lector no debería esperar encontrarse una descripción detallada de cada una de las tareas genéricas

que conforman el ejemplo, ni una especificación real de las partes en las que se puede dividir. Quizás entonces cabría preguntarse el porqué del calificativo *real*, cuando después de la afirmación anterior parece que este término pierde parte de su significado.

He utilizado este adjetivo por dos razones fundamentales:

- *Primero, porque la estructura y complejidad de las tareas son propias* de las aplicaciones reales. De hecho, observando las características de la aproximación macroscópica, lo que importa conocer no son los detalles internos de cada una de las tareas, sino más bien su carácter general y grado de complejidad. De esta manera, la esencia del ejemplo se mantiene, y es suficiente para calibrar las técnicas de estimación y particionamiento.
- Segundo, porque el grado de exactitud utilizado está dentro, o incluso en un nivel superior, de los límites estándar que catalogan el concepto de *realidad* observados en la literatura relacionada actual. Así, es común encontrarse con aproximaciones que proponen casos en los que se garantiza el total realismo del ejemplo, y que luego se limitan a una interconexión de tareas que contienen una serie de parámetros de no muy clara procedencia.

C.1.1. Importancia de las aplicaciones de tratamiento de imagen.

Hoy en día, uno de los tipos de procesos más actuales y exigentes en cuanto a los requerimientos técnicos necesitados para su funcionamiento lo constituye el conjunto de aplicaciones basadas en el tratamiento de imágenes.

Esto se debe a la gran presencia de utilidades que usan secuencias de vídeo en tiempo real como parte básica de su funcionamiento,

especialmente aprovechando las cada día más mejoradas características que proporcionan las redes.

De esta manera, se constituye una necesidad impulsada por el entorno para el desarrollo de esta serie de aplicaciones, por lo que se está haciendo prácticamente indispensable la utilización de técnicas de diseño automático que sean capaces de tratarlos de una forma eficiente.

Debido a su complejidad, fundamentalmente asociada a la gran potencia de cálculo requerida, las técnicas tradicionales de desarrollo manual han probado ser insuficientes en la calidad de los resultados proporcionados, a no ser que se utilice un largo tiempo de diseño.

De igual manera, el uso de metodologías de Codiseño tradicionales en las que se introduzcan procesos de estimación y particionamiento clásicos, está abocado a la falta de eficiencia, debido a la gran complejidad temporal que ofrecen.

Por lo tanto, parece *a priori* interesante aplicar los nuevos conceptos desarrollados en este tipo de aplicaciones, a fin de estudiar si los efectos positivos manifestados sobre casos experimentales se siguen aún manteniendo en estas circunstancias.

C.1.2. Estructura del codificador.

El estándar H.261 presentado fue creado en 1991, y posteriormente refinado en sucesivos ajustes, por la *International Telecommunication Union* [Itu93]. Engloba una especificación de codificación para la transmisión de secuencias de vídeo digital. El rango de aplicaciones a las que va destinado son aquellas que demandan una frecuencia baja de trabajo de 64 Kbits/segundo. Esta frecuencia limitada es la que diferencia a este estándar de otros más extendidos hoy en día, como el MPEG, que requiere unos anchos de banda de entre 0.9 y 1.5 Mbits/segundo.

La labor fundamental de este codificador es el agrupamiento de la gran cantidad de información asociada a los formatos de imagen digital, de tal forma que su transmisión resulte mucho menos costosa, en términos de tiempo, de lo que en circunstancias naturales sería.

El formato sobre el que trabaja es el denominado YCbCr, asociado al sistema PAL. Su característica fundamental es la división de la información contenida en la imagen en distintos bloques, que codifican las llamadas características de *luminancia*, *crominancia en fase* y *crominancia cuadrática* [Hung93]. Con estos parámetros se define unívocamente el objeto, trabajando en un espacio simétrico, pero más fácil de manipular que el tradicional sistema RGB.

El esquema de tareas que representa a este codificador se ofrece en la Figura C.1. La unidad de información que fluye a través de este diagrama, y sobre el que se aplican las distintas transformaciones, es el *macrobloque*, que se define de una manera estándar como la agrupación de cuatro bloques de luminancia y dos de crominancia.

A continuación, se explica, en términos generales, la semántica dentro del proceso general de cada una de las tareas que aparecen en el diagrama.

C.1.3. Modelado y determinación de las características intrínsecas.

La característica fundamental del codificador es la eliminación máxima de toda información redundante, a fin de que la velocidad de transmisión sea lo más alta posible. Para ello, cada una de las imágenes es codificada mediante un desplazamiento respecto a la anterior. De esta manera, y debido a que entre imágenes consecutivas de una misma secuencia no suelen existir grandes diferencias, la información requerida es de un tamaño reducido.

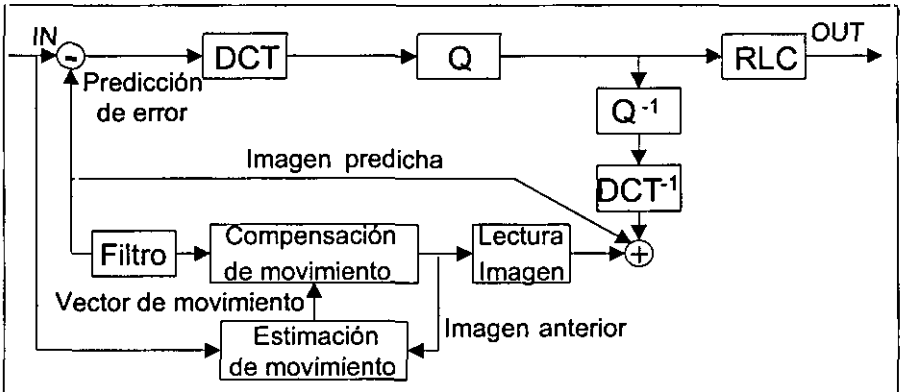


Figura C.1

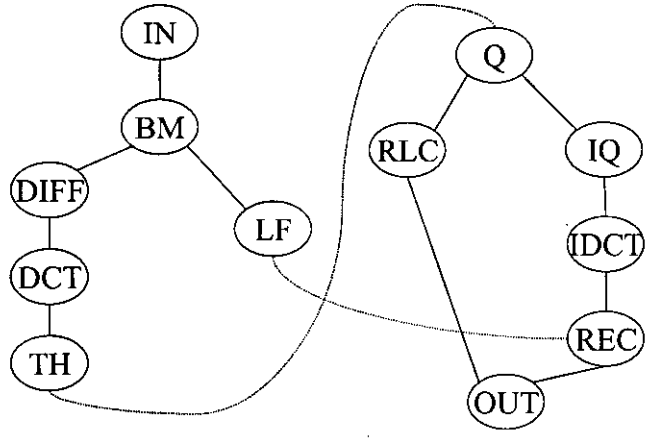
Esquema de tareas del codificador H.261

Primeramente, los procesos de *estimación y compensación del movimiento*, tienen como función el cálculo de la diferencia entre la imagen actual y la anterior, que se encuentra almacenada en memoria.

El módulo de *filtro* (F) obedece a un comportamiento típico especificado por su nombre, el cual actúa sobre la predicción de la actual imagen a partir de la anterior, como fruto de los procesos comentados anteriormente.

Esta imagen predicha se compara con la real, y se genera un vector de error relacionado, al que se le aplica el proceso asociado a la *Transformada Discreta de Coseno* (DCT). Esta operación tiene por objeto concentrar la información que se encuentra a su entrada, y ofrece como resultado un dato similar en el que algunos de sus coeficientes se han agrupado, y se han creado otros nulos.

Después, se realiza el proceso de *Cuantificación* (Q), en el que se prosigue con una pérdida controlada de información. Esta pérdida tiene por objeto hacer nulos un mayor número de coeficientes, sin que se pierda la semántica de reconstrucción de la imagen.



Nodo	IN	BM	DIFF	LF	DCT	TH	Q	RLC	IQ	IDCT	REC	OUT
Impl.	1	1	3	3	4	2	4	2	3	3	2	1

Figura C.2

Grafo de Codiseño asociado y número de implementaciones

Finalmente, se lleva a cabo la operación de *Codificación en función de la longitud* (RLC), la cual se realiza de una forma entrópica, para disminuir el tamaño de la información, y se transmite de tal manera que exista un máximo número de ceros consecutivos. Esto hace que el dato se compacte en gran medida, lo que facilita su posterior tratamiento a una alta velocidad.

Los módulos indicados por Q^{-1} y DCT^{-1} representan las operaciones inversas de Q y DCT respectivamente.

A partir de este diagrama de bloques, se procede a la extracción del grafo de Codiseño asociado que lo representa, y que se puede observar en la Figura C.2. No existe una equivalencia biunívoca entre tareas y nodos, a fin de facilitar los procesos posteriores. De todos modos, esto corresponde a

una decisión de elección de la granularidad, sujeta a criterios variables. En este caso, se ha escogido la propuesta en [BITT98] [TeBT97].

Las diferencias producidas se basan en la agrupación de la *estimación y compensación de movimiento* sobre el nodo de *reconocimiento de bloque* (BM), y la separación de la *cuantificación* en los nodos de *cálculo de umbral* (TH) y *cuantificación básica* (Q).

Asimismo, el módulo de resta corresponde al nodo DIFF, el de suma al nodo REC, y las inversas Q^{-1} y DCT^{-1} se han etiquetado como IQ e IDCT. IN y OUT representan los procesos de adquisición y almacenamiento de datos.

El siguiente paso realizado ha consistido en el modelo y determinación de los parámetros asociados a todos estos nodos. Para cada uno de ellos, se ha elegido el número de implementaciones reflejado en la tabla de la Figura C.2.

Aquí, como se comentó anteriormente, no se ha seguido un análisis real de la especificación de cada una de las tareas, sino que, aprovechando el generador de grafos automático, se han caracterizado utilizando unos parámetros adecuados que simulen su comportamiento.

De tal manera, se han escogido unos límites de nodo de entre 15 y 18 etapas de control, con un número de unidades funcionales por etapa comprendido entre 2 y 10.

El modelo software se ha caracterizado mediante un factor de forma de 20, con una desviación del ±10%.

Con todas estas especificaciones, el grafo de Codiseño queda especificado, y se procede a detallar los resultados experimentales obtenidos tras la aplicación de las técnicas de estimación y particionamiento propuestas.

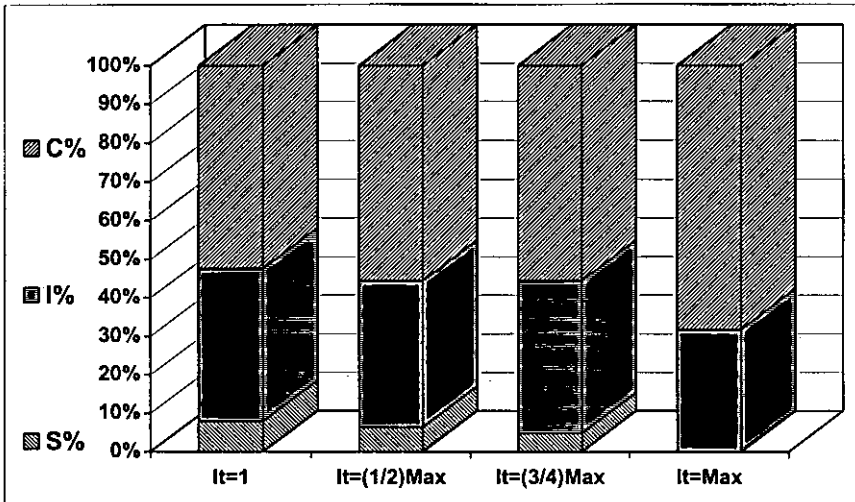


Figura C.3

Media comparativa de los resultados del particionamiento

C.2 Resultados experimentales obtenidos.

Los procesos experimentales considerados y los estudios posteriores son exactamente los mismos que los ofrecidos en §7.2 y §7.3. Se muestran los resultados separados en el módulo de particionamiento y el módulo de estimaciones.

C.2.1. Proceso de particionamiento.

El particionamiento se ha llevado a cabo en un rango de restricciones temporales comprendidas entre los valores de 97 y 534, que corresponden a los tiempos asociados a la solución más rápida y más lenta posible. En total, se han considerado 10 puntos intermedios.

Para cada uno de estos puntos de restricción, se ha procedido a llevar a cabo el proceso de particionamiento mediante la técnica estándar y utilizando las cuatro alternativas de agrupamiento (it=1, it=½Max, it=¾Max,

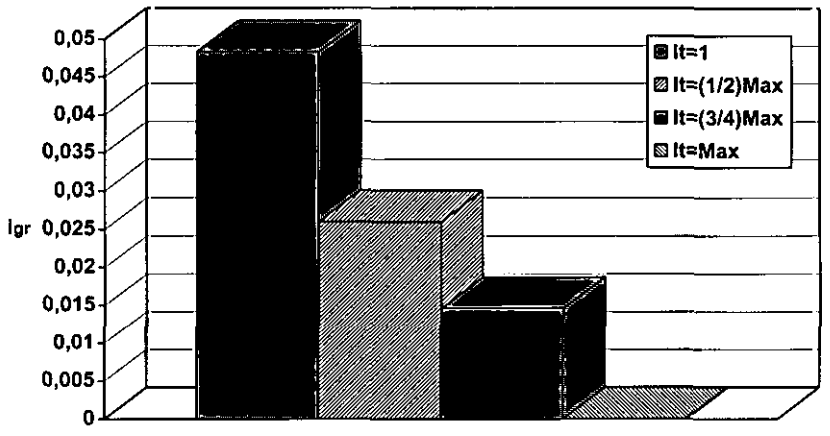


Figura C.4

Media del índice de impacto

it=Max). De esta manera, se comparan los porcentajes de soluciones mejores, peores e iguales obtenidas por estas frente a las alcanzadas por aquella.

Los resultados se pueden ver agrupados en la Figura C.3. Cuando se considera una única iteración de agrupamiento, se observa que se obtiene un porcentaje de soluciones mejores muy considerable, frente a una pequeña muestra de soluciones peores (*C*%=52.38%, *I*%=39.68%, *S*%=7.94%).

Cuando se procedió a liberar la mitad de las comunicaciones agrupadas, el número de soluciones mejores aumentó ligeramente, con una disminución también pequeña del número de soluciones peores (*C*%=55.56%, *I*%=38.10%, *S*%=6.34%).

En el siguiente paso, al liberar tres cuartos de las comunicaciones, se mantiene esta tendencia, con un estancamiento del número de soluciones mejores (*C*%=55.56%, *I*%=39.68%, *S*%=4.76%).

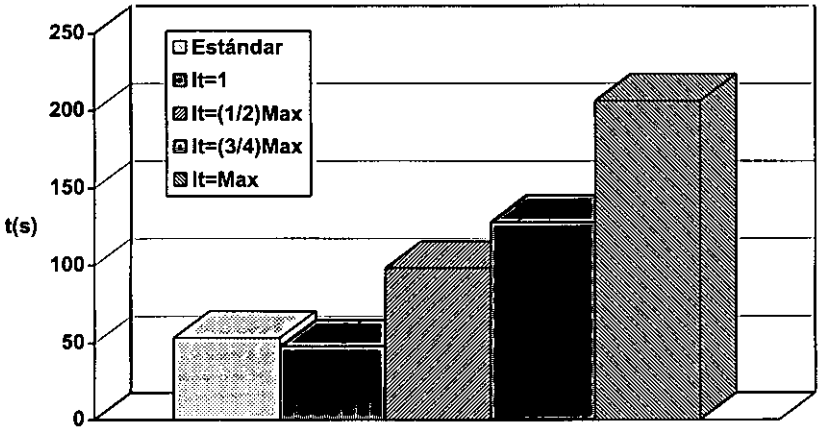


Figura C.5
Media del tiempo de proceso

Finalmente, y como es obvio, al liberar la totalidad de las comunicaciones se elimina todo el número de soluciones peores (*C*%=68.25%, *I*%=31.75%, *S*%=0.00%).

Respecto al estudio del definido *índice de impacto*, que mide la relación entre los costes extras introducidos por las soluciones peores y los costes ganados por las soluciones mejores, los resultados también son satisfactorios, como se puede comprobar en la Figura C.4.

En el peor de los casos, con una única iteración de agrupamiento, el índice toma un valor de 0.0483, lo que indica la poca repercusión de las soluciones peores encontradas. Este índice disminuye hasta 0.0260, posteriormente pasa a 0.0146, hasta llegar a hacerse nulo.

Finalmente, se ha procedido a efectuar el estudio de tiempos de proceso, cuyos resultados aparecen en la Figura C.5. El particionamiento estándar consume de media 53.5273 segundos, y cada una de las

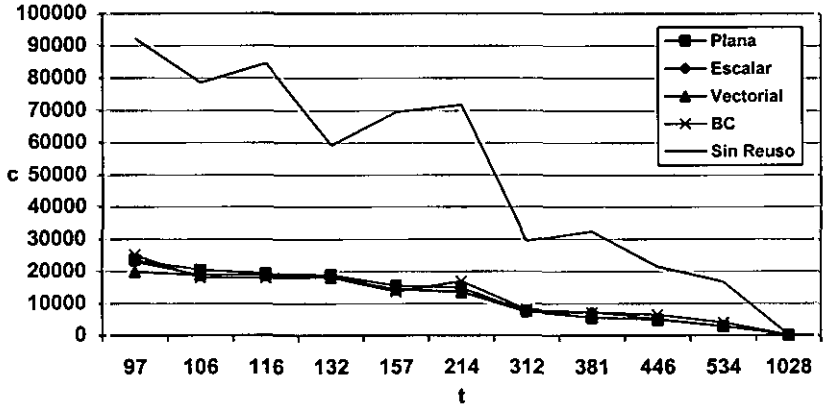


Figura C.6

Resultados comparativos de la técnica sin reuso

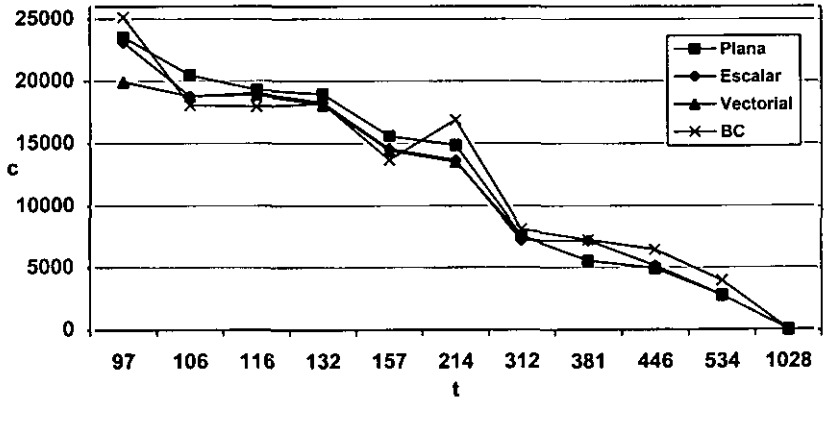


Figura C.7

Resultados comparativos con el Behavioral Compiler

aproximaciones con agrupamiento 47.9173, 98.2476, 128.0401, 206.2584 segundos respectivamente.

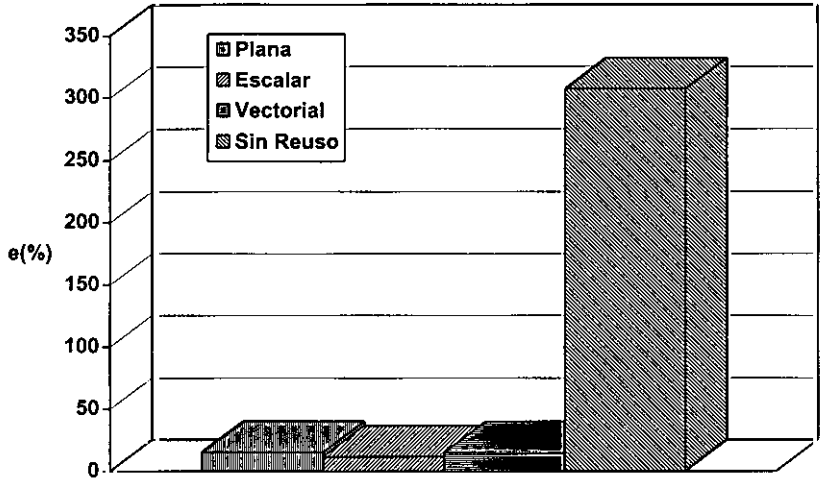


Figura C.8

Media del error de precisión

Se observa que todos los resultados ofrecidos en esta sección siguen la tendencia y corroboran las conclusiones extraídas en el capítulo anterior.

C.2.2. Proceso de estimación.

Después del proceso de particionamiento, se ha procedido a la estimación de las soluciones asociadas a las distintas restricciones temporales comentadas anteriormente, mediante las cinco técnicas utilizadas en §7.2 (plana, escalar, vectorial, sin reuso y *Behavioral Compiler).*

Los resultados obtenidos se observan en la Figura C.6. Aquí se comprueba que, debido a la mayor complejidad de este ejemplo, la técnica sin reuso se desvía más que en casos anteriores. Esto es obvio, ya que la importancia de este factor cobra más trascendencia en este caso.

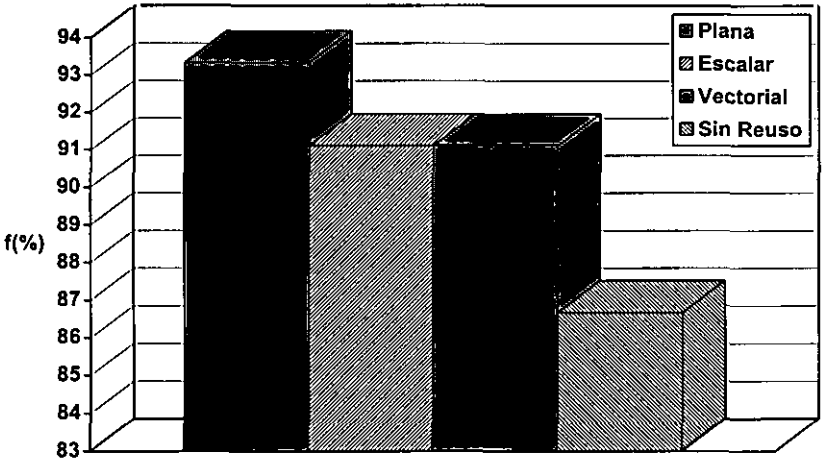


Figura C.9

Media de la fidelidad

El resto de las técnicas se pueden ver amplificadas en la Figura C.7. La conclusión que se extrae es que todas siguen una tendencia muy parecida y ajustada, por lo que es posible corroborar el hecho de que a mayor complejidad, las distintas aproximaciones tienden a converger.

El primer estudio realizado sobre estos resultados es el del error de precisión. Los valores hallados se pueden observar en la Figura C.8. De esta manera, se tiene que el valor relacionado con la técnica plana es del 15.2370%, el asociado a la escalar es del 11.5143% y el relativo a la vectorial es del 14.8447%. Por otro lado, se obtiene que la técnica sin reuso produce un error del 306.9847%, el cual es mucho mayor que en cualquier experimento desarrollado hasta ahora, debido sin duda al aumento de la complejidad.

En cuanto al estudio de fidelidad, los resultados aparecen en la Figura C.9. La técnica plana ofrece un valor relacionado del 93.33%, mientras que

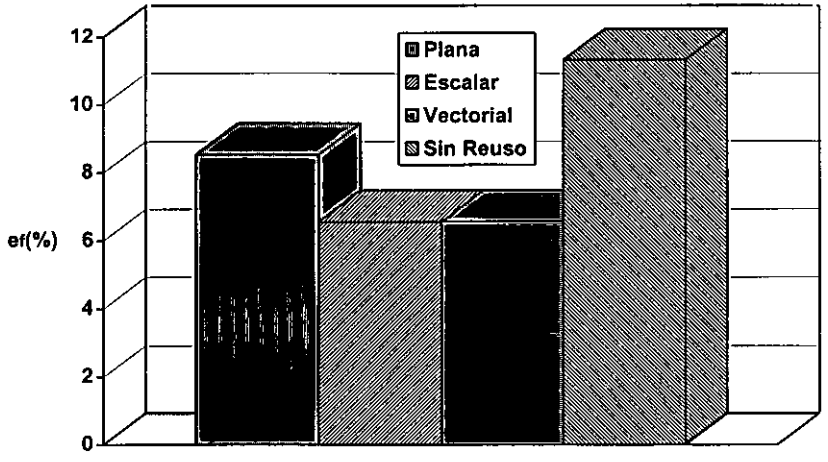


Figura C.10

Media del error asociado a la fidelidad

ambas escalar y vectorial obtienen una fidelidad del 91.11%. Finalmente, la técnica sin reuso tiene un valor del 86.67% asociado a este parámetro.

El estudio del error asociado a la fidelidad se muestra en la Figura C.10. Aquí, la desviación cometida por las técnicas es del 8.5359%, 6.5496%, 6.5496% y 11.3262%, para las aproximaciones plana, escalar, vectorial y sin reuso respectivamente.

Por último, se ha procedido a la medición de los tiempos de proceso consumidos, y han sido comparados con el asociado al *Behavioral Compiler*. Todos ellos pueden verse en la Figura C.11, excepto este último, cuyo valor de 3429 segundos excede en gran manera el orden de magnitud asociado al resto de las técnicas, las cuales consumen un tiempo medio de 0.0045, 0.1789 y 0.4399 segundos para las técnicas plana, escalar y vectorial respectivamente.

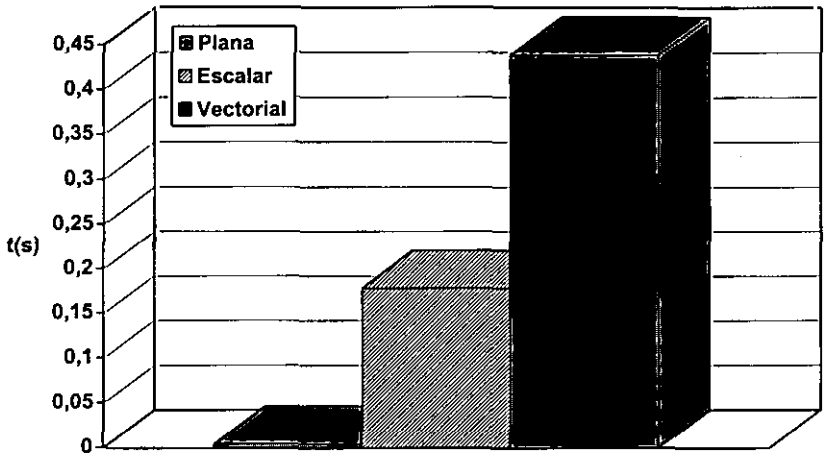


Figura C.11

Media del tiempo de proceso

C.3 Conclusiones acerca del proceso experimental.

Se pueden extraer varias conclusiones a la vista del proceso experimental llevado a cabo en este apéndice:

Los resultados obtenidos corroboran los previamente hallados en §7, por lo que las técnicas de estimación y particionamiento quedan ampliamente contrastadas.

Se sigue manteniendo la eficacia para problemas de una alta complejidad, que son el escollo fundamental para el asentamiento definitivo de las técnicas de Codiseño.

La técnica tradicional sin reuso obtiene un error de estimación mucho mayor que los hallados previamente, por lo que se prueba que el aumento de la complejidad es incompatible con las técnicas clásicas.

Se cumple uno de los objetivos prioritarios marcados al comienzo del trabajo: el mantenimiento del tiempo de diseño lo más bajo posible. Así, se ve que este tiempo consumido por el *Behavioral Compiler* se dispara enormemente, mientras que los relacionados con las técnicas presentadas logran mantener su bajo orden de magnitud.

Referencias.

[AFSS98] A. Allara, W. Fornaciari, F. Salice, D. Sciuto, "A Model for System-Level Timed Analysis and Profiling", Proc. Design, Automation and Test in Europe, DATE'98.

[AlKa95] C. Alpert, A. Kahng, "Recent directions in netlist partitioning: a survey", Integration, the VLSI journal 19, 1995.

[AlLR96] G. Al-Hayek, Y. Le-Traon, C. Robach, "Considering Test Economics in the Process of Hardware/Software Partitioning", Proc. EUROMICRO'96.

[AVVH98] A. Antón, E. Villar, D. de Vries, S. Heemstra de Groot, "Design and functional description of a sender and receiver for ATM Adaptation Layer Protocols", Proc. Design of Circuits and Integrated Systems, DCIS'98.

[BaGa97] S. Bakshi, D. Gajski, "Hardware/Software Partitioning and Pipelining", Proc. Design Automation Conference, DAC'97.

[BaSa97] F. Balarin, A. Sangiovanni-Vincentelli, "Schedule Validation for Embedded Reactive Real-Time Systems", Proc. Design Automation Conference, DAC'97.

[BeEO95] T. Benner, R. Ernst, A. Österling, "Scalable Performance Scheduling for Hardware-Software Cosynthesys", Proc. European Design Automation Conference, EURODAC'95.

[BeLR97] J. Bergé, O. Levia, J. Rouillard (Ed.), "Hardware/Software Co-Design and Co-Verification", Kluwer, 1997.

[BHJL89] T. Bui, C. Heigham, C. Jones, T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms", Proc. Design Automation Conference, DAC'89.

[BISH94] N. Binh, M. Imai, A. Shiomi, N. Hikichi, J. Sato, "Extension of Instruction Set Design for Pipelined Architecture in PEAS-I System", DA Symposium, IPSJ, Japón, vol. 94, #5, Agosto 1994, pp. 211-216.

[BLMS98] F. Balarin, L. Lavagno, P. Murthy, A. Sangiovanni-Vincentelli, "Scheduling for Embedded Real-Time Systems", IEEE Design & Test of Computers, Enero-Marzo 1998, pp. 71-82.

[BITT98] T. Blicke, J. Teich, L. Thiele, "System-Level Synthesis Using Evolutionary Algorithms", Design Automation for Embedded Systems, 3, 1998, pp. 23-58.

[CaWi95] R. Camposano, J. Wilberg, "Embedded System Design", Journal of Design Automation for Embedded Systems, vol.1, #1, 1995, pp. 5-50.

[CHMP98] J. Calvez, D. Heller, F. Muller, O. Pasquier, "A Programmable Multi-Language Generator for CoDesign", Proc. Design, Automation and Test in Europe, DATE'98.

[DaJh98] B. Dave, N. Jha, "CASPER: Concurrent Hardware-Software Co-Synthesis of Hard Real-Time Aperiodic and Periodic

Specifications of Embedded System Architectures", Proc. Design, Automation and Test in Europe, DATE'98.

[Dave99] B. Dave, "CRUSADE: Hardware/Software Co-Synthesis of Dynamically Reconfigurable Heterogeneous Real-Time Distributed Embedded Systems", Proc. Design, Automation and Test in Europe, DATE'99.

[DeKM90] G. De Micheli, D. Ku, F. Mailhot, "The OLYMPUS Synthesis System for Digital Design", Design & Test Magazine, vol. 3, Octubre 1990, pp. 37-53.

[DeSh98] A. Dean, J. Shen, "Hardware to Software Migration with Real-Time Thread Integration", Proc. EUROMICRO'98.

[DiJh97] R. Dick, N. Jha, "MOGAC : A Multiobjective Genetic Algorithm for the Co-Synthesis of Hardware-Software Embedded Systems", Proc. International Conference on Computer-Aided Design, ICCAD'97.

[EKPD98a] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, P. Pop, "Process Scheduling for Performance Estimation and Synthesis of Hardware/Software Systems", Proc. EUROMICRO'98.

[EKPD98b] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, P. Pop, "Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems", Proc. Design, Automation and Test in Europe, DATE'98.

[ElPD94] P. Eles, Z. Peng, A. Doboli, "VHDL System-Level Specification and Partitioning in a Hardware/Software Co-Synthesis Environment", Proc. Third International Workshop on Hardware/Software Codesign, Septiembre 1994.

[EPKD96] P. Eles, Z. Peng, K. Kuchcinski, A. Doboli, "System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search", Design Automation for Embedded Systems, 2, 1996, pp. 5-32.

[ErHB93] R. Ernst, J. Henkel, T. Benner, "Hardware-Software Cosynthesis for Microcontrollers", IEEE Design and Test of Computers, Diciembre 1993, pp. 64-75.

[Erns98] R. Ernst, "Codesign of Embedded Systems: Status and Trends", IEEE Design and Test of Computers, Abril-Junio 1998, pp. 45-53.

[FiMa82] C. Fiduccia, R. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", Proc. Design Automation Conference, DAC'82.

[GaVN94] D. Gajski, F. Vahid, S. Narayan, "A System-Design Methodology: Executable-Specification Refinement", Proc. European Design Automation Conference, EDAC'94.

[GDWL92] D. Gajski, N. Dutt, A. Wu, S. Lin, "High-Level Synthesis. Introduction to Chip and System Design", Kluwer, 1992.

[GGVN93] D. Gajski, J. Gong, F. Vahid, S. Narayan, "The SpecSyn Design Process and Human Interface", UC Irvine TR #93-3, 1993.

[GhKL99] A. Ghosh, J. Kunkel, S. Liao, "Hardware Synthesis from C/C++", Proc. Design, Automation and Test in Europe, DATE'99.

[GrMa98] J. Grode, J. Madsen, "A Unified Component Modeling Approach for Performance Estimation in Hardware/Software Codesign", Proc. EUROMICRO'98.

[GuBS93] A. Gupta, W. Birmingham, D. Siewiorek, "Automating the Design of Computer Systems", Trans. On Computer-Aided Design of Integrated Circuits and Systems, vol.12, #4, Abril 1993.

[GuDe97] R. Gupta, G. De Micheli, "Constrained software synthesis for embedded applications", Journal of Systems Architecture, 43, 1997, pp. 557-586.

[Gupt95] R. Gupta, "Co-Synthesis of Hardware and Software for Digital Embedded Systems", Ed. Kluwer, 1995.

[GVNG94] D. Gajski, F. Vahid, S. Narayan, J. Gong, "Specification and Design of Embedded Systems", Ed. Prentice-Hall, 1994.

[HaHK95] L. Hagen, D. Huang, A. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", Proc. European Design Automation Conference, EURODAC'95

[HaRo97] W. Hardt, W. Rosenstiel, "Prototyping of Tightly Coupled Hardware/Software Systems", Design Automation for Embedded Systems, Volumen 2, #3/4, Mayo 1997.

[HeEr97] J. Henkel, R. Ernst, "A Hardware/Software Partitioner using a dynamically determined Granularity", Proc. Design Automation Conference, DAC'97.

[HEHB94] J. Henkel, R. Ernst, U. Holtmann, T. Benner, "Adaptation of Partitioning and High-Level Synthesis in Hardware/Software Cosynthesis", Proc. International Conference on Computer-Aided Design, ICCAD'94.

[HeHE94] D. Herrmann, J. Henkel, R. Ernst, "An Approach to the Adaptation of Estimated Cost Parameters in the COSYMA

System", Proc. Third International Workshop on Hardware/Software Codesign, Septiembre 1994.

[HeOl97] R. Helaihel, K. Olukotum, "Java as a Specification Language for Hardware-Software Systems", Proc. International Conference on Computer-Aided Design, ICCAD'97.

[HiBo97] K. Hines, G. Borriello, "Dynamic Communication Models in Embedded System Co-Simulation", Proc. Design Automation Conference, DAC'97.

[HoEr93] U. Holtmann, R. Ernst, "Speculative Computation for Coprocessor Synthesis", Proc. International Conference on Computer Design, ICCD'93.

[HuDa97] X. Hu, J. D'Amborsia, "Hardware-Software Partitioning for Real-Time Embedded Systems", Design Automation for Embedded Systems, Volumen 2, #3/4, Mayo 1997.

[HuDe94a] I. Huang, A. Despain, "Generating Instruction Sets and Microarchitectures from Applications", Proc. International Conference on Computer-Aided Design, ICCAD'94.

[HuDe94b] I. Huang, A. Despain, "Synthesis of Instruction Sets for Pipelined Microprocessors", Proc. Design Automation Conference, DAC'94.

[HuJe98] J. van den Hurk, J. Jess, "System Level Hardware/Software Co-design. An Industrial Approach", Kluwer, 1998.

[Hung93] A. Hung, "PVRG-P64 Codec 1.1", Information of the Portable Video Research Group, Stanford University, 1993.

[IsJe95] T.-B. Ismail, A. Jerraya, "Synthesis Steps and Design Models for Codesign", Computer Magazine, Febrero 1995.

[IsOJ94] T.-B. Ismail, K. O'Brien, A. Jerraya, "Interactive System-Level Partitioning with PARTIF", Proc. European Design Automation Conference, EDAC'94.

[Itu93] International Telecommunication Union, "Video Codec for Audiovisual Services at px64 Kbits. ITU-T Recommendation H.261", Helsinki 1993.

[Kala95] A. Kalavade, "System-Level Codesign of Mixed Hardware-Software Systems", Tesis Doctoral, Universidad de California, Berkeley, Septiembre 1995.

[KaLe93] A. Kalavade, E. Lee, "A Hardware/Software Codesign Methodology for DSP Applications", IEEE Design and Test of Computers, Septiembre 1993, pp. 16-28.

[KaLe94] A. Kalavade, E. Lee, "A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem", Proc. Third International Workshop on Hardware/Software Codesign, Septiembre 1994.

[KaLe95] A. Kalavade, E. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping and Implementing-Bin Selection", Sixth International Workshop on Rapid Systems Prototyping, Chapel Hill, Junio 1995.

[KaLe97] A. Kalavade, E. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling, and Implementation-Bin Selection", Design Automation for Embedded Systems, vol.2, #2, Marzo 1997, pp. 125-164.

[KaSu97] A. Kalavade, P. Subrahmanyam, "Hardware/Software Partitioning for Multi-function Systems", Proc. International Conference on Computer-Aided Design, ICCAD'97.

[KeLi70] B. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell System Technical Journal, Febrero 1970.

[KiGV83] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by Simulated Annealing", Science 220: 671-680, 1983.

[Knap96] D. Knapp, "Behavioral Synthesis. Digital System Design Using the Synopsys Behavioral Compiler", Ed. Prentice-Hall, 1996.

[KuGa92] D. Ku, D. Gajski, "Relative Scheduling Under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits", IEEE Trans. On Computer-Aided Design, vol.11, #6, Junio 1992.

[LCLS96] M. López, C. Carreras, J. López, L. Sánchez, "Coarse Grain Partitioning for Hardware-Software Codesign", Proc. EUROMICRO'96.

[Leup99] R. Leupers, "Exploiting Conditional Instructions in Code Generation for Embedded VLIW Processors", Proc. Design, Automation and Test in Europe, DATE'99.

[Liem97] C. Liem, "Retargetable Compilers for Embedded Core Processors. Methods and Experiences in Industrial Applications", Kluwer, 1997.

[LiMW95] Y. Li, S. Malik, A. Wolfe, "Efficient microarchitecture modeling and path analysis for real-time software", 16th. IEEE Real-Time Systems Symposium, Diciembre 1995, pp. 298-307.

[LiMW96] Y. Li, S. Malik, A. Wolfe, "Cache modeling for real-time software: Beyond direct mapped instruction caches", 17th. IEEE Real-Time Systems Symposium, Diciembre 1996, pp. 254-263.

[Lin98] B. Lin, "Efficient Compilation of Process-Based Concurrent Programs without Run-Time Scheduling", Proc. Design, Automation and Test in Europe, DATE'98.

[LKHC95] L.-T. Liu, M.-T. Kuo, S.-C. Huang, C.-K. Cheng, "A Gradient Method on the Initial Partition of Fiduccia-Mattheyses Algorithm", Proc. International Conference on Computer-Aided Design, ICCAD'95.

[LMTJ98] J. Leijten, J. Meerbergen, A. Timmer, J. Jess, "Stream Communication between Real-Time Tasks in a High-Performance Multiprocessor", Proc. Design, Automation and Test in Europe, DATE'98.

[LolL98] M. López, C. Iglesias, J. López, "A Knowledge-based System for Hardware-Software Partitioning", Proc. Design, Automation and Test in Europe, DATE'98.

[LoLo98] M. López, J. López, "Knowledge Based Hardware-Software Partitioning of Electronic Systems", en "Advanced techniques for embedded systems design and test", J. López, R. Hermida, W. Geisselhardt (Ed.), Kluwer, 1998.

[Lope99] M.L. López Vallejo, "Métodos para la distribución de funcionalidad entre recursos hardware y software en el diseño de sistemas heterogéneos", Tesis Doctoral, Universidad Politécnica, Madrid, Enero 1999.

[LVSV97] A. López, M. Veiga, P. Sánchez, E. Villar, "Embedded system specification in ADA", Proc. Design of Circuits and Integrated Systems, DCIS'97.

[MaDJ97] G. Marchioro, J. Daveau, A. Jerraya, "Transformational Partitioning for Co-Design of Multiprocessor Systems", Proc.

International Conference on Computer-Aided Design, ICCAD'97.

[Maes96] J.A. Maestro, "New Methodologies for Hardware-Software Codesign Partitioning to Avoid High Communication Overhead", Proc. 4th. BELSIGN Workshop, Octubre 1996.

[MaGA95] M. Martonosi, A. Gupta, T. Anderson, "Tuning Memory Performance in Sequential and Parallel Programs", IEEE Computer, Abril 1995, pp. 32-40.

[MaMC96] J.A. Maestro, D. Mozos, D. Camacho, "Una técnica de particionamiento para reducir la sobrecarga por comunicaciones en Codiseño", Proc. Design of Circuits and Integrated Systems, DCIS'96.

[MaMH99] J.A. Maestro, D. Mozos, R. Hermida, "The Heterogeneous Structure Problem in Hardware/Software Codesign: A Macroscopic Approach", Proc. Design, Automation and Test in Europe, DATE'99.

[MaML97] S. Malik, M. Martonosi, Y. Li, "Static Timing Analysis of Embedded Software", Proc. Design Automation Conference, DAC'97.

[MaMM98] J.A. Maestro, D. Mozos, H. Mecha, "A Macroscopic Time and Cost Estimation Model Allowing Task Parallelism and Hardware Sharing for the Codesign Partitioning Process", Proc. Design, Automation and Test in Europe, DATE'98.

[MaMo96] J.A. Maestro, D. Mozos, "Estudio de métodos de particionamiento para el problema de Codiseño", Technical Report #35.96, Dpto. de Arquitectura de Computadores, Universidad Complutense, Madrid, Junio 1996.

[MaMo97] J.A. Maestro, Daniel Mozos, "Resusability and Parallelism Effects on the Partitioning Process for a Codesign Environment", Proc. Design of Circuits and Integrated Systems, DCIS'97.

[MaMo98a] J.A. Maestro, D. Mozos, "A Solution to the Hardware Sharing Problem in Codesign Using a Macroscopic Approach", Proc. Design of Circuits and Integrated Systems, DCIS'98

[MaMo98b] J.A. Maestro, D. Mozos, "Time and Cost Trade-off Exploration Using Hardware Sharing and Parallelism in the Codesign Process", Proc. 7th. BELSIGN Workshop, Mayo 1998.

[MaMS98] J.A. Maestro, D. Mozos, J. Septién, "A Grouping Partitioning Technique with Automatic Criterion Selection for the Codesign Process", Proc. EUROMICRO'98.

[MDBD95] J. Mira, A. Delgado, J. Boticario, F. Díez, "Aspectos básicos de la Inteligencia Artificial", Ed. Sanz y Torres, 1995.

[MFTS96] H. Mecha, M. Fernández, F. Tirado, J. Septién, D. Mozos, K. Olcoz, "A Method for Area Estimation of Data-Path in High Level Synthesis", IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, vol.15, #2, Febrero 1996.

[MGKP97] J. Madsen, J. Grode, P. Knudsen, M. Peterson, A. Haxthausen, "LYCOS: the Lyngby Co-Synthesis System", Design Automation for Embedded Systems, vol.2, #2, Marzo 1997, pp. 195-236.

[MKBS99] R. Maestre, F. Kurdahi, N. Bagherzadeh, H. Singh, R. Hermida, M. Fernández, "Kernel Scheduling in Reconfigurable Computing", Proc. Design, Automation and Test in Europe, DATE'99.

[MoDe97] V. Mooney, G. De Micheli, "Real Time Analysis and Priority Scheduler Generation for Hardware-Software Systems with a Synthesized Run-Time System", Proc. International Conference on Computer-Aided Design, ICCAD'97.

[MRKD98] V. Mooney, D. Ruspini, O. Khatib, G. De Micheli, "Hardware-Software Run-Time Systems and Robotics: A Case Study", Proc. EUROMICRO'98.

[NaGa94] S. Narayan, D. Gajski, "Synthesis of System-Level Bus Interface", Proc. European Design Automation Conference, EDAC'94.

[NaVG92] S. Narayan, F. Vahid, D. Gajski, "System Specification with the SpecCharts Language", IEEE Design and Test, Diciembre 1992.

[Nebe99] W. Nebel, "JAVA, VHDL-AMS, ADA or C for System Level Specifications?", Proc. Design, Automation and Test in Europe, DATE'99.

[Newe81] A. Newell, "The Knowledge Level", AI Magazine, summer 1981, pp. 1-20.

[NiMa96] R. Niemann, P. Marwedel, "Hardware/Software Partitioning using Integer Programming", Proc. European Design and Test Conference, EDTC'96.

[NiMa97] R. Niemann, P. Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming", Design Automation for Embedded Systems, vol.2, #2, Marzo 1997, pp. 165-194.

[NiMa98] R. Niemann, P. Marwedel, "Synthesis of Communicating Controllers for Concurrent Hardware/Software Systems", Proc. Design, Automation and Test in Europe, DATE'98.

[ObIJ93] K. O'Brien, T.-B. Ismail, A. Jerraya, "A Flexible Communication Modelling Paradigm For System-Level Synthesis", Proc. International Workshop on Hardware-Software Co-Design, Cambridge, Octubre 1993.

[OnOJ98] M. O'Nils, J. Öberg, A. Jantsch, "Grammar Based Modelling and Synthesis of Device Drivers and Bus Interfaces", Proc. EUROMICRO'98.

[PLCS97] C. Passerone, L. Lavagno, M. Chiodo, A. Sangiovanni-Vincentelli, "Fast hardware/software co-simulation for virtual prototyping and trade-off analysis", Proc. Design Automation Conference, DAC'97.

[PlGM98] P. Plöger, H. Günter, E. Moser, "An Industrial Case Study in HW-SW Co-Design using CASTLE", en "Advanced techniques for embedded systems design and test", J. López, R. Hermida, W. Geisselhardt (Ed.), Kluwer, 1998.

[PlWi94] P. Ploeger, J. Wilberg, "Tutorial for CASTLE: Design Example Video Compression System", GMD-Report, Proyecto SYDIS, 1994.

[PlWi95] P. Ploeger, J. Wilberg, "A Codesign Example using CASTLE Tools", Proc. Workshop on Design Methodology for Microelectronics, Eslovaquia, Septiembre 1995.

[PoWo95] M. Potkonjak, W. Wolf, "Cost Optimization in ASIC Implementation of Periodic Hard-Real Time Systems using

Behavioral Synthesis Techniques", Proc. International Conference on Computer-Aided Design, ICCAD'95.

[RuMS96] P. Rupérez, D. Mozos, J. Septién, "Dnet: An Internal Representation Structure for a HW/SW Codesign System", Proc. EUROMICRO'96.

[SAHN94] J. Sato, A. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, M. Imai, "PEAS-I: A Hardware/Software Codesign System for ASIP Development", Trans. On IEICE, vol. E77-A, #3, Marzo 1994, pp. 483-491.

[Shob98] M. El Shobaki, "Verification of Embedded Real-Time Systems Using Hardware/Software Co-simulation", Proc. EUROMICRO'98.

[StWo97] J. Staunstrup, W. Wolf (Ed.), "Hardware/Software Co-Design: Principles and Practice", Kluwer, 1997.

[SuHa98] W. Sung, S. Ha, "Optimized Timed Hardware Software Cosimulation without Roll-back", Proc. Design, Automation and Test in Europe, DATE'98.

[TeBT97] J. Teich, T. Blicke, L. Thiele, "An Evolutionary Approach to System-Level Synthesis", Proc. 5th International Workshop on Hardware/Software Codesign, Braunschweig, Marzo 1997.

[ThSV94] M. Theibinger, P. Stravers, H. Veit, "Castle: An Interactive Environment for HW-SW Co-Design", Proc. Third International Workshop on Hardware/Software Codesign, Septiembre 1994.

[TVVV95] P. Tabuenca, E. Villar, H. Veit, H. Vierhaus, "Hardware/Software Codesign Environment based on the CASTLE and FIRES tools: Use of C and VHDL as Specification Languages", GMD-Report, Proyecto SYDIS, 1995.

[VaCJ97] C. Valderrama, A. Changuel, A. Jerraya, "Virtual Prototyping for Modular and Flexible Hardware-Software Systems", Design Automation for Embedded Systems, Volumen 2, #3/4, Mayo 1997.

[VaGa95a] F. Vahid, D. Gajski, "Closeness metrics for system-level functional partitioning", Proc. European Design Automation Conference, EURODAC'95.

[VaGa95b] F. Vahid, D. Gajski, "Incremental Hardware Estimation During Hardware/Software Functional Partitioning", IEEE Trans. On VLSI Systems, vol. 3, #3, Septiembre 1995, pp. 459-464.

[VaGG93] F. Vahid, J. Gong, D. Gajski, "A Hardware-Software Partitioning Algorithm for Minimizing Hardware", Technical Report ICS-93-38, Septiembre 1993.

[VaGG94] F. Vahid, J. Gong, D. Gajski, "A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning", Proc. European Design Automation Conference, EURODAC'94.

[Vahi91] F. Vahid, "A Survey of Behavioral-Level Partitioning System", Technical Report ICS-91-71, Octubre 1991.

[VaLe97] F. Vahid, T. Le, "Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning", Design Automation for Embedded Systems, vol.2, #2, Marzo 1997, pp. 237-261.

[VeLi97] S. Vercauteren, B. Lin, "Hardware/Software Communications and System Integration for Embedded Architectures", Design Automation for Embedded Systems, Volumen 2, #3/4, Mayo 1997.

[VeVV99] S. Vercauteren, J. Van Der Steen, D. Verkest, "Combining Software Synthesis and Hardware/Software Interface Generation to Meet Hard Real-Time Constraints", Proc. Design, Automation and Test in Europe, DATE'99.

[VIJK94] M. Voss, T.-B. Ismail, A. Jerraya, K.-H. Kapp, "Towards a Theory for Hardware/Software Codesign", Proc. Third International Workshop on Hardware/Software Codesign, Septiembre 1994.

[ViVe98] E. Villar, M. Veiga, "Embedded system specification", en "Advanced techniques for embedded systems design and test", J. López, R. Hermida, W. Geisselhardt (Ed.), Kluwer, 1998.

[VPGS98] J. Voeten, P. van der Putten, M. Geilen, M. Stevens, "System Level Modelling for Hardware/Software Systems", Proc. EUROMICRO'98.

[WoFr92] W. Wolf, E. Frey, "Tutorial on Embedded System Design", Proc. International Conference on Computer Design, ICCD'92.

[WoJa95] W. Wong, R. Jain, "PARAS: System-Level Concurrent Partitioning and Scheduling", Proc. International Conference on Computer-Aided Design, ICCAD'95.

[Wolf94] W. Wolf, "Hardware-Software Co-Design of Embedded Systems", Proceedings of the IEEE vol.82, #7, Julio 1994.

[WuLa94] Y. Wu, J. Larus, "Static Branch Frequency and Program Profile Analysis", Micro-27, Proceedings of the 27th Annual International Symposium on MicroArchitecture, Noviembre 1994.

[YEBH93] W. Ye, R. Ernst, T. Benner, J. Henkel, "Fast Timing Analysis for Hardware-Software Co-Synthesys", Proc. International Conference on Computer Design, ICCD'93.