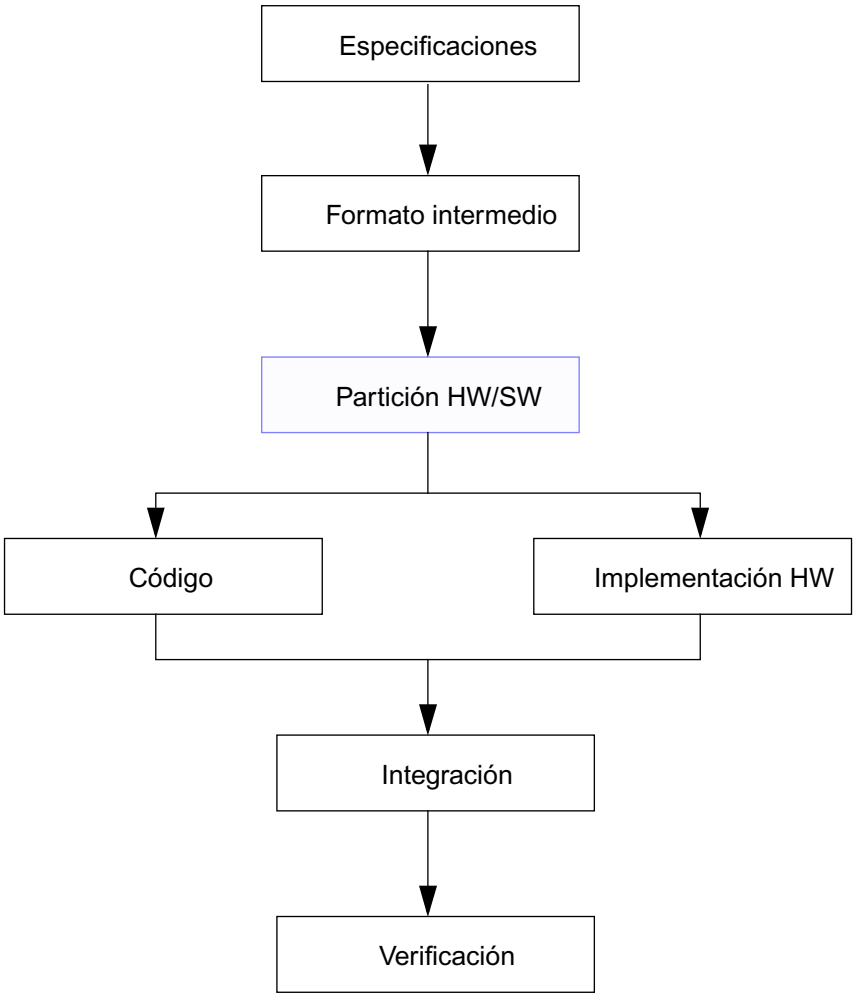
Tema III

Particionado HW/SW

El siguiente paso en el flujo de diseño será la partición hardware/software. En este tema trataremos este paso, tras el cual tendremos separadas las operaciones implementadas vía software y hardware.



3.1. Introducción

La partición hardware/software es una de las tareas principales del diseño de sistemas empotrados o del codiseño HW/SW. Esta tarea se puede definir de la siguiente forma:

> La partición HW/SW es la tarea que determinará la funcionalidad implementada vía hardware y vía software.

Con ella se tratará de aprovechar las ventajas hardware y software de la arquitectura en la cual se integrará el sistema. Las funciones que se implementan vía software reducirán el tiempo de diseño, mientras que aquellas implementadas vía hardware aumentarán las prestaciones. Por lo tanto, se tiende hacia un sistema con poco coste y mayores prestaciones; no obstante, para comprobar que el particionado elegido sea bueno, se debe cumplir un compromiso entre los costes y las prestaciones. Esta elección se obtiene siguiendo el esquema mostrado en la figura 3.1. En primer lugar debemos especificar el sistema utilizando un lenguaje y un modelo de computación. Este sistema será simulado y los estimadores nos indicarán los atributos del sistema, tanto hardware como software. Los atributos principales son la latencia, el área hardware y el espacio en memoria. Una vez que tenemos los atributos, se introducen en la función de coste para comprobar si se cumplen los compromisos entre costes y prestaciones. En función de dicha función (valga la redundancia), un algoritmo cambiará el particionado para tratar de cumplir los compromisos.

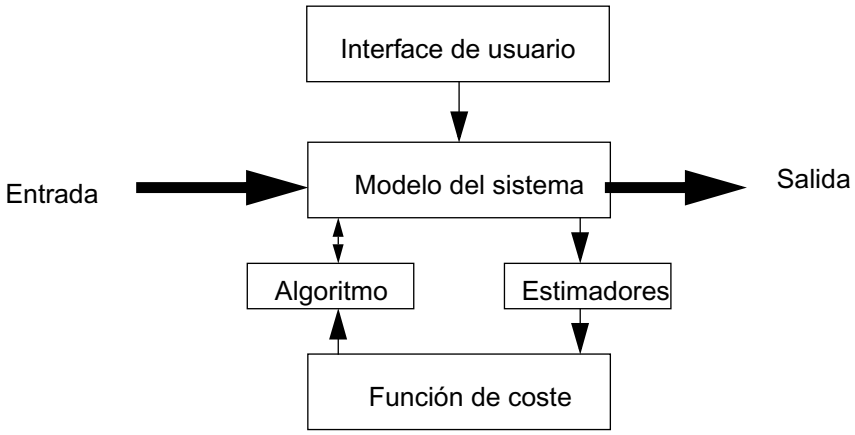


Figura 3.1.- Esquema para elegir el particionado HW/SW.

A la hora de abordar este problema hay que considerar una serie de factores como son el tipo de especificación, la granularidad, la arquitectura destino y el tipo de aplicación.

Con el tipo de especificación obtendremos un modelo para nuestro sistema. No debemos confundirlo con el modelo de computación, el cual nos va a determinar cómo se va a realizar la simulación. Para especificar el sistema podemos utilizar grafos o un lenguaje. De estas dos opciones elegiremos un lenguaje ya que tienen un tratamiento más sencillo. Además haremos uso de varios lenguajes: uno para la descripción hardware; y otro para la descripción software.

La granularidad es la propiedad que indica el tamaño de las primitivas (objetos) con las que se trabaja. Básicamente diferenciaremos dos tipos de granularidad:

- Fina, cuando se trata de instrucciones y operaciones
- Gruesa, cuando se trabaja con tareas, funciones y bloques.

Así se puede optar por mantener una única granularidad, y mantener las mismas primitivas para el diseño completo; o bien utilizar diferentes granularidades para partes distintas del sistema. Esta última elección sería la más deseable, ya que se podría utilizar una granularidad más fina para las partes más críticas, y otra más gruesa para aquellas menos críticas. Esta situación se vuelve compleja a la hora de implementar el algoritmo, ya que debería ser capaz de tratar los dos tipos de granularidad.

La arquitectura destino nos va a indicar los elementos que van a ser utilizados en la implementación final del sistema. La arquitectura clásica se muestra en la figura 3.2. En esta arquitectura podemos distinguir cuatro grandes tipos de elementos: interfaces de entrada/salida, para la comunicación con el medio exterior; un sistema de memoria, para almacenar los datos necesarios para su funcionamiento; un procesador, para ejecutar las funciones implementadas vía software; y coprocesador, para ejecutar las funciones implementadas vía hardware. Este punto se tratará en mayor detalle en el apartado siguiente.

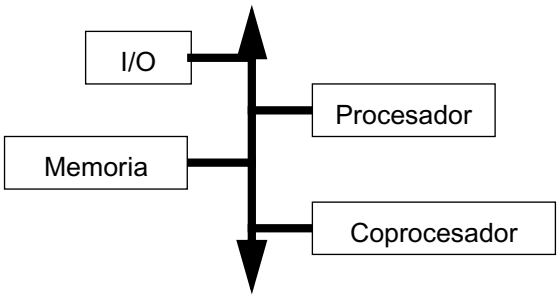


Figura 3.2.- Arquitectura clásica de un sistema empotrado.

Finalmente hay que considerar el tipo de aplicación, ya que de ella va a depender las prestaciones necesarias, y por lo tanto la partición idónea para el sistema.

En la tabla 3.1 se muestra una comparativa de los métodos de particionado más usados en materia de investigación.

El método propuesto por Ernst utiliza un modelo basado en grafos de control y datos, donde cada nodo corresponde a operaciones elementales (sentencias en una extensión de C con capacidad de concurrencia). La función de coste para determinar la idoneidad de la partición es derivada en tres aspectos:

- En la parte software se realziará una tarea de profiling, de tal forma que se buscarán los cuellos de botella que pueden ser eliminados moviendo algunas operaciones a la parte hardware.
- En la parte hardware se buscará la similitud entre operaciones, tanto en control, en datos como en operadores. De esta forma, aquellas operaciones con un determinado grado de similitud se mantendrán en la parte hardware, y aquellas que no lo tengan pasarán a la parte software.

Autor	Modelo	Granularidad	Función de coste	Algoritmo
Ernst	CDFG	Operación	-Profiling (sw) -Síntesis y similitud (hw) -Costes de comunicación	-Simulación (interno) -Mano (externo)
Olokutun	HDL	Tarea	-Profiling (sw) -Síntesis (hw)	-Kernighan y Lin
Kumar	Basado en conjuntos	Tarea	-Profiling	-Progr. matemática
Hu	Lista de tareas	Tarea	-Profiling -Análisis de esquema	-branch and bound
Vahid	DFG acíclico	Operación	-Profiling (sw) -Costes de procesador (hw) -Costes de comunicación	-Progr. entera-lineal
Barros	HDL	Operación	-Similitud -Concurrencia/secuencialidad	-Clustering
Kalavade	DFG acíclico	Operación	-Esquemalibilidad	-Heurístico con adelanto
Steinhausen	CDFG	Operación	-Profiling	-Mano
Ben Ismail	SDL	Tarea	-	-Mano
Chou	Diagrama temporal	Operación	-Tiempo (sw) -Área (hw)	-Mínimo corte

Tabla 3.1. Métodos de partición.

- Por último se estimará la sobrecarga debida a la comunicación necesaria entre las partes hardware y software. La estimación se aproxima al número de datos intercambiados entre ambas partes, asumiendo un mecanismo de comunicación simple de memoria mapeada.

Por último, el algoritmo utilizado para cambiar el particionado se basa en un doble bucle. El primer bucle utiliza la simulación, con un rápida estimación de la ganancia derivada por el cambio de particionado. El segundo bucle utiliza la síntesis para obtener una estimación más refinada.

El método propuesto por Olokutun utiliza un lenguaje de descripción de hardware para obtener el modelo del sistema, en el que el tratamiento mínimo serán los bloques o tareas. Esto permite usar herramientas de síntesis para estimar el coste en recursos de hardware y el profiling de un simulador para código compilado para la estimación de coste en los recursos de software. El particionado se realiza junto con el scheduling, ya que la meta final consiste en minimizar el tiempo de respuesta en el contexto de utilizar emulación para acelerar la simulación. Una partición inicial se obtiene clasificando los bloques según sean sintetizables o no y si la sobrecarga de comunicación justifica una implementación hardware. Esta partición inicial es mejorada utilizando un procedimiento iterativo como el de Kernighan y Lin.

El método propuesto por Kumar considera el particionado de una forma muy general y abstracta. El modelado del sistema se realiza con una representación compleja basada en conjuntos. El coste se determina básicamente por profiling. El sistema diseñado se representa por cuatro conjuntos: funciones software disponibles; recursos hardware; comunicación entre los elementos; y las funciones que necesitan ser implementadas. El proceso de partición es seguido por una descomposición de cada función en conjunto de instrucciones virtuales, seguido por el diseño de un implementación para el conjunto usando los recursos disponibles, y por último se llevará a cabo una fase de evaluación.

El método propuesto por Hu aborda el problema de elegir un conjunto de procesadores sobre los cuales se pueden ejecutar un conjunto de tareas mientras que se cumplen una serie de restricciones. También utilizan una formulación matemática, pero proveen una solución óptima a través de un procedimiento branch-and-bound. El coste de la partición software es estimado como unos límites inferior (a través de varios refinamientos de la suma de los tiempos de computación de las tareas dividido entre el periodo de las tareas) y superior (a través de un análisis de tasas monotónico) sobre la utilización del procesador. El proceso de optimización utiliza estos límites para explorar el espacio de diseño, mientras busca asignaciones óptimas para las funciones respetando las limitaciones temporales. Otros criterios de optimización que pueden ser incluidos son el coste de hardware y la expansibilidad.

El método propuesto por Vahid realiza un particionado basado en grafos de una especificación de granularidad variable. El lenguaje utilizado es SpecCharts, un modelo jerárquico en el que cada hoja es un estado de una descripción como diagramas ASM jerárquica. Estos estados pueden contener procesos VHDL de complejidad arbitraria, escritos en un estilo de especificación de alto nivel. La estimación de coste se realiza a nivel de hoja, en la que se asigna un número estimado de puertos de entrada/salida, un área estimada (que puede cambiar si dicho recurso ya se ha incluido en otras hojas) y un tiempo de ejecución estimado. El modelo de coste es adjuntado a un grafo en el que cada nodo es una hoja y los arcos representan dependencias de control y dato. Algoritmos clásicos de particionado y agrupamiento son entonces aplicados, tras los cuales se realiza un proceso de refinamiento. Durante el refinamiento, cada partición es sintetizada para obtener una mejor estimación de área y tiempos, y los nodos periféricos son movidos entre las particiones para reducir el coste. El coste de cada partición es simplemente la suma ponderada del área, puertos, número de chips y medidas de satisfacción de restricciones.

El método propuesto por Barros usa una representación de granularidad fina basado en grafos, en los que cada unidad se corresponde con una instrucción simple del lenguaje de especificación UNITY. Ellos agrupan las unidades de acuerdo con una serie de criterios (similitud entre unidades, concurrencia, secuencialidad, exclusión mutua y vectorización de una secuencia de asignaciones relacionadas), de tal forma que minimiza el coste de cortes en la estructura de grupos, y por lo tanto, mejora el agrupamiento considerando oportunidades de pipeline, localizaciones hechas en la etapa previa, y ahorro de costes debido a la distribución de recursos.

El método propuesto por Kalavade utiliza un grafo de dependencia acíclica. Este grafo se deriva partiendo de un grafo de flujo de datos, se realiza un mapeo simultáneo de cada nodo a recursos software o hardware, y se esquematiza la ejecución de las tareas. La aproximación es heurística, y puede dar una solución aproximada a grandes sistemas. Para guiar el proceso de búsqueda, utiliza información del camino crítico y la adecuabilidad de un nodo para recursos hardware o software.

El método propuesto por Steinhausen describe un entorno completo de cosíntesis, en el que una representación de grafo de flujo de datos y control es derivado de una matriz de especificaciones de alto nivel (tanto en lenguaje software como hardware). Este grafo es particionado a mano basado en los resultados de profiling, y entonces mapeado en una arquitectura que puede contener microprocesadores de propósito general, ASIPs y ASICs. Un aspecto interesante en este método es que la arquitectura no es fija, sino que es dirigida por la descripción estructural definida por el usuario. La síntesis hardware se realiza mediante una herramienta comercial; mientras que la síntesis software se realiza mediante un compilador (independiente de la arquitectura software) desarrollado por Hoogerbrugge.

El método propuesto por Ben Ismail empieza en una descripción en el lenguaje SDL, que será traducida a una interconexión jerárquica de procesos comunicantes. Los procesos pueden ser unidos y divididos, y la jerarquía se puede cambiar dividiendo, moviendo y agrupando subunidades. La secuencia de estas operaciones es llevada a cabo por el usuario.

El método propuesto por Chou describe un algoritmo especializado basado en esquemas para el particionado de interfaces. El algoritmo está basado en un modelo de grafos derivado de un diagrama temporal formalizado. Los nodos representan eventos de bajo nivel en la especificación de la interface. Los arcos representan limitaciones, y pueden ser derivadas de enlaces causales o ser añadidas durante el proceso de particionado. La función de coste es el tiempo para el software y el área para el hardware. El algoritmo está basado en un procedimiento de migración de grupos de mínimo corte aplicado al grafo para reducir la congestión (definida como el software que es requerido para producir eventos más rápidamente que el procesador puede hacerlo, lo cual implica la necesidad de algún hardware de asistencia).

3.2. Arquitecturas Hardware/Software

En este apartado nos centraremos en la arquitectura del sistema empotrado, que clásicamente se identifica con la mostrada en la figura 3.2. En ella nos centraremos en los dos componentes fundamentales (procesador y coprocesador), que distinguirán los dos tipos de implementaciones: software y hardware.

3.2.1. Arquitecturas hardware

Dentro de las arquitecturas hardware incluiremos a los coprocesadores necesarios, que serán aquellos elementos necesarios para aumentar las prestaciones del sistema completo. Estos elementos serán implementados generalmente en un ASIC o en una FPGA.

Las arquitecturas hardware consideradas se pueden dividir en tres grandes grupos: controladores, caminos de datos y controladores más camino de datos.

Un controlador se corresponderá con la implementación de típica de los sistemas secuenciales a partir de un diagrama de estados. Por lo tanto, el esquema de esta arquitectura es la mostrada en la figura 3.3, donde podemos distinguir las funciones lógicas de salida y de próximo estado, y el banco de registros en los que se almacena el estado actual del controlador.

Por contra, la arquitectura de camino de datos se basa en el desarrollo de bloques de una determinada funcionalidad (más compleja que puertas lógicas y biestables), los cuales serán

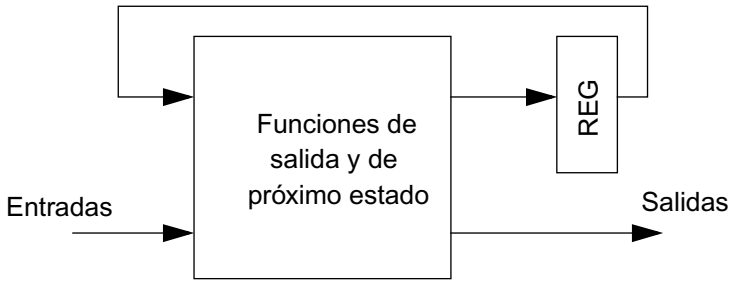


Figura 3.3.- Arquitectura de un controlador

controlados a través del cableado del sistema. Estos bloques podrán ser diseñados utilizando diferentes arquitecturas dependiendo de las prestaciones requeridas. En la figura 3.4 podemos ver dos implementaciones diferentes de un filtro digital, en las que una de ellas presenta una mayor regularidad y la otra una mayor velocidad.

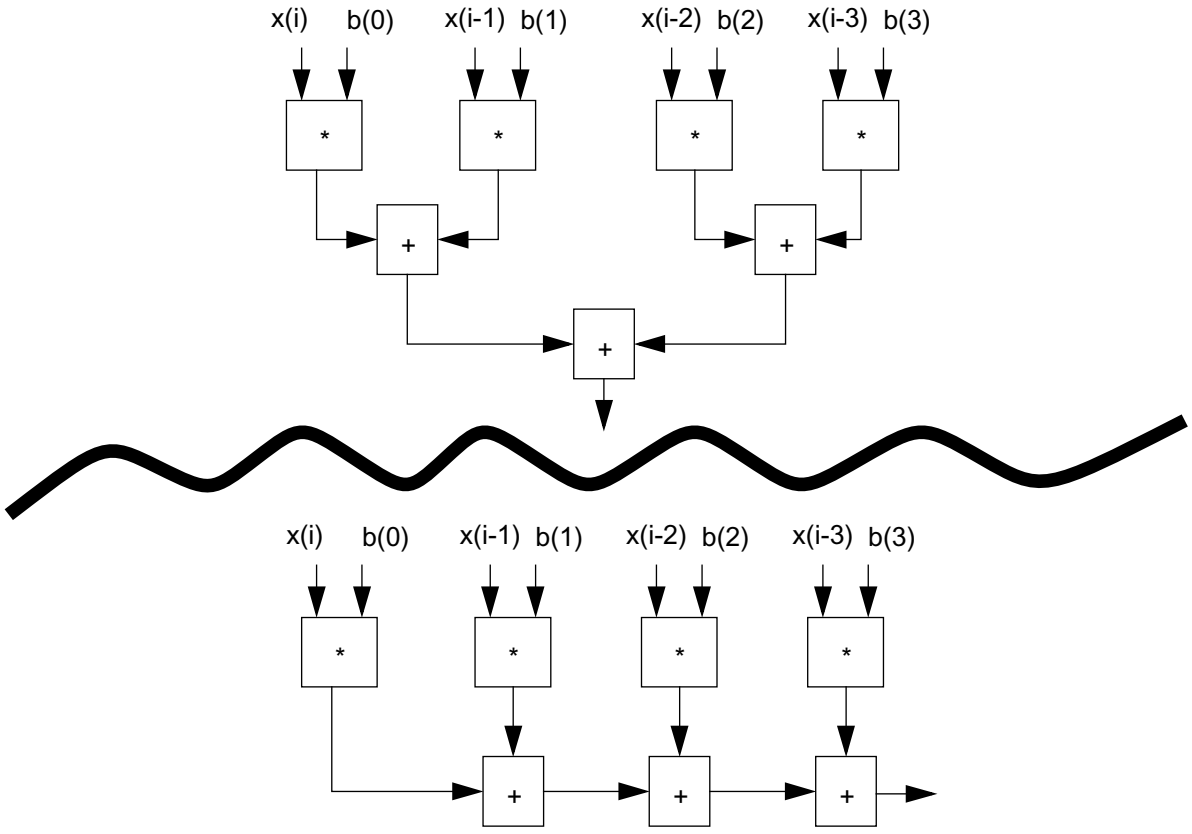


Figura 3.4.- Implementaciones de un posible bloque de un camino de datos.
$F = x(i)\cdot b(0) + x(i-1)\cdot b(1) + x(i-2)\cdot b(2) + x(i-3)\cdot b(3)$

También existe una arquitectura que mezcla ambas posibilidades, es decir, existen unos bloques de camino de datos (con funcionalidad única o con posibilidad de programabilidad) que son controlados por un controlador en lugar de por cableado. Esta arquitectura sería la que se obtendría desde una descripción de transferencia de registro (a partir de un diagrama ASM). Un esquema de esta arquitectura se muestra en la figura 3.5.

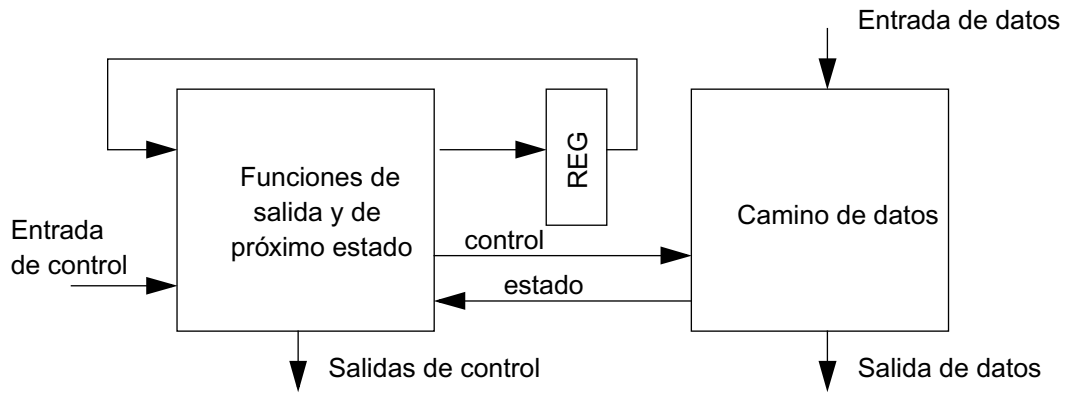


Figura 3.5.- Arquitectura de un controlador más camino de dato

3.2.2. Arquitecturas software

En los dispositivos software se englobarán todos aquellos bloques cuya función puede ser definida a través de un programa software, como pueden ser DSP, microcontroladores y microprocesadores. Si detallamos más, podemos encontrar los siguientes elementos software: DSP (Digital Signal Processor), ASIP (Application Specific Instruction Processor), ISP (Instruction Set Processor).

En el caso de los DSP, su arquitectura es muy similar a la de un procesador general, pero con unos bloques de procesado específicos para el tratamiento de la señal digital. Entre las principales características que tienen los DSP podemos encontrar las siguientes:

- ALU más un multiplicador paralelo
- Memoria y registros eficientes
- Múltiples modos de direccionamiento

Los ASIP son procesadores de carácter específico para la aplicación en la que se vayan a utilizar. Entre las principales características de este tipo de procesadores podemos encontrar las siguientes:

- Características arquitecturales
  - Instrucciones especializadas
  - Modos de direccionamiento
  - Manipulación de bits
  - Conversión de tipos de datos
  - Control
- Características del conjunto de instrucciones
  - Pocas instrucciones
  - Microcódigo (instrucciones de un solo ciclo) y macrocódigo (instrucciones de varios ciclos)
  - Instrucciones de campo fijo o codificadas
  - Operaciones entre registros
  - Estructuras de registros generales o especificas

En el caso de los ISP podemos encontrarnos diferentes tipos de arquitecturas, como pueden ser CISC (Complex Instruction Set Computer), RISC (Reduced Instruction Set Computer) y VLIW (Very Long Instruction Word). Todos ellos siguen alguna de las arquitecturas clásica de un procesador genérico (Harvard o Von Newmann), las cuales se muestran en la figura 3.6. La diferencia entre las versiones CISC y RISC son básicamente unas unidades de control y aritmético-lógicas más o menos complejas. En el caso de la arquitectura VLIW, la unidad aritmético-lógica se diseña utilizando varias unidades de un tamaño de datos normal colocadas en paralelo.

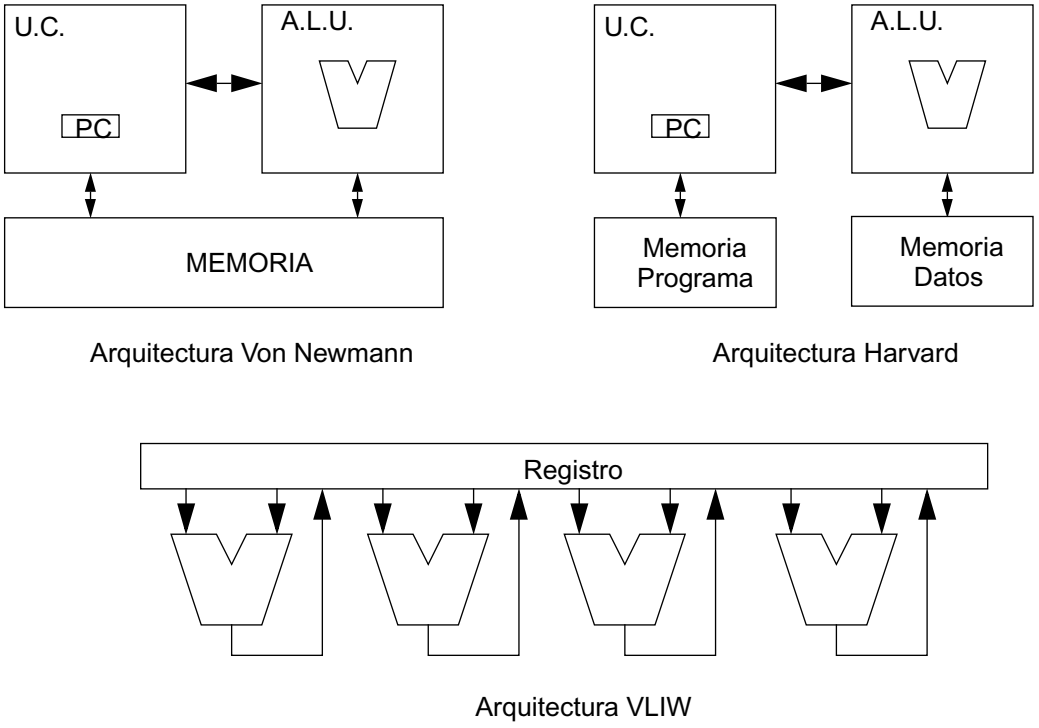


Figura 3.6.- Arquitecturas clásicas de procesadores, y arquitectura VLIW.

A pesar de poder utilizar dispositivos generales, lo más óptimo es la utilización de dispositivos de aplicación específica como DSP o ASIP. En este caso, existen algunas técnicas de especialización para mejorar las prestaciones del sistema. Estas técnicas las podemos diferenciar entre técnicas de componentes (especificas para cada uno de los componentes del procesador), de sistemas (específicas para el funcionamiento del sistema completo) y de comunicaciones. Entre las técnicas de componentes podemos distinguir las siguientes:

- Conjunto de instrucciones especializados, con lo cual se pueden encadenar las operaciones (sin necesidad de ejecutar nuevas instrucciones), y se puede explotar la concurrencia en la sobrecarga de control (las operaciones para capturar una nueva instrucción, realizadas de forma paralela a alguna operación de datos).
- Camino de datos y unidades funcionales especializadas, hecho que está en perfecta sintonía con la técnica anterior.
- Estructura de memoria especializada, en el sentido de que tanto el número como el tamaño de los bancos de memoria y puertos no sean estándares, sino que sean adecuados a la aplicación deseada.

- Conectividad y control especializado, hecho que está en perfecta sintonía con las dos primeras. A través de ella se puede conseguir un flujo de control mixto: cableado y programado, reduciendo los accesos a memoria (por ejemplo).

Dentro de las técnicas de sistema podemos encontrar las siguientes:

- Descomposición del control, a través de un flujo de control mixto (cableado y programado), o técnicas de control de componentes especializadas
- Descomposición de datos, a través de su distribución sobre bloques de memoria especializados
- Flujo de control simple, en el que el procesador central inicia la operación que es completada por el coprocesador (o coprocesadores). En el caso de utilizar varios coprocesadores para una misma operación, estos se deben comunicar para conocer el estado de cada uno de los bloques.
- Tramas de control múltiples
- Conocimiento de las estructuras y secuencias de datos permiten la especialización del sistema de memoria.

Dentro de las técnicas de comunicación podemos encontrar las siguientes:

- Comunicaciones punto a punto, en las que el canal es único para cada elementos conectados. Se puede realizar tanto con como sin buffering, para acelerar la transmisión efectiva de datos.
- Comunicaciones a través de un bus genérico, en las que todos los elementos se conectan al mismo bus y su controlador decide hacia que elementos deben ir los datos. Al igual que en el caso anterior, entre el bus y los elementos pueden existir bufferes o no.

3.3. Técnicas de partición Hardware/Software

Una vez que ya hemos comentado las posibilidades arquitecturales de un sistema empotrado, pasaremos a comentar los principales algoritmos de partición. Entre ellos podemos encontrar el recocido simulado, la programación lineal y los algoritmos genéticos; además también existen adaptaciones de soluciones a otros problemas como son la migración de grupos y clustering; y técnicas de inteligencia artificial.

3.3.1. Técnica de recocido simulado (simulated annealing)

Esta técnica se basa en el proceso físico de recocido para la obtención de cristales de mineral, en el cual se realiza una perturbación al material (generalmente un cambio de temperatura) para llevar al material a un estado de mínima energía. Éste es un proceso recursivo que acabará cuando se alcance el mínimo global, el cual se alcanzará en el momento de la cristalización.

Como se puede comprobar el proceso del recocido puede ser perfectamente implementado utilizando el esquema de la figura 3.1, donde el modelo es el estado del material, los atributos es la energía, la función de coste es la función de mínima energía y el algoritmo es la perturbación del sistema.

Para que el algoritmo alcance la solución óptima en el menor número de pasos posibles, hay algunos elementos que deben tratarse muy bien:

- La función de energía (coste) debe reflejar perfectamente la calidad de una solución
- El conjunto de perturbaciones (formado por las diferentes configuraciones del sistema)
- El esquema de *enfriamiento* (reducción del coste), que será el que decida cuando se ha alcanzado el mínimo y cuál es la siguiente perturbación para procesar.

El algoritmo que se sigue es el siguiente:

```
Construir partición inicial X^actual:=(Hw_0,Sw_0)
Inicializar temperatura T:=Ti
Repetir
   for i:=1 to Tl do
      Generar aleatoriamente una partición vecina a la actual X'
      Calcular el cambio de la función de coste ΔC:= C(X')-C(X^actual)
      if ΔC <= 0 then X^actual:= X'
      else
         Generar q:=random(0,1)
         if q < e^-ΔC/T then X^actual:=X' end if
      end if
   end for
   Elegir una nueva temperatura T:= α·T
Hasta que se encuentre el criterio de parada (generalmente una temperatura
umbral)
Devuelve la solución correspondiente a la mínima función de coste
```

Como principales componentes del algoritmo, cabe destacar que busca nuevas soluciones de forma aleatoria, sin intentar obtener a priori soluciones óptimas. Esta nueva solución se corresponderá con el cambio de partición de un nodo, siempre y cuando no viole ninguna de las limitaciones de diseño. Esta situación influye en las principales ventajas y desventajas del algoritmo.

Entre las principales ventajas que tiene este método, podemos encontrar las siguientes:

- Escapa de mínimos locales.
- Es independiente de la solución de partida

Mientras que entre los principales inconvenientes podemos encontrar:

- Tiene una tiempos de ejecución muy elevados
- Cuando hay muchos parámetros, es muy difícil de ajustar
- No se garantiza un mínimo global

Existe una búsqueda guiada de soluciones para reducir el tiempo de ejecución en la que se reduce la selección aleatoria de nodos a los vecinos del último nodo seleccionado (entendiéndose como nodo vecino a aquellos que están directamente conectados entre sí).

Existe otro método de optimización basado en el recocido simulado aprovechando la ventaja de la búsqueda histórica del siguiente movimiento. Este nuevo método se denomina búsqueda tabú (tabu search). En este caso, el algoritmo utilizado es el siguiente:

```
Construir partición inicial X^actual:=(Hw_0,Sw_0)
Comienzo:
   for each partición vecina a la partición inicial do
      Calcular el cambio de la función de coste ΔC_k:= C(X_k)-C(X^actual)
   end for
   for each ΔC_k < 0 en orden creciente de ΔC_k do
      if not tabu(X_k) or tabu_aspirante(X_k) then
         X^actual:=X_k
         goto aceptar
      end if
   end for
   for each partición vecina a la partición inicial do
      Calcular ΔC'_k:= ΔC_k + penalidad(X_k)
   end for
for each ΔC'_k en orden creciente do
      if not tabu(X_k) then
         X^actual:=X_k
         goto aceptar
      end if
   end for
   Generar X^actual por el desarrollo del movimiento menos tabu
Aceptar:
   if nº de iteraciones desde la mejor solución previa < Nr_f_b then
      goto comienzo
   end if
   if restarts < Nr_r then
      Generar partición inicial X^actual considerando las frecuencias
      goto comienzo
   end if
   Devuelve la partición correspondiente a la mínima función de coste
```

En este caso, se buscan soluciones de menor coste que no estén en la lista de los movimientos tabús (prohibidos). En caso de no encontrar ninguno, se busca una partición buena basada en una función de penalidad. La lista de los movimientos prohibidos es actualizada añadiendo los movimientos inversos a los realizados en la partición actualizada.

3.3.2. Migración de grupos (mínimo corte)

Esta técnica es una adaptación de un método heurístico, basado en la experiencia. El procedimiento consiste en agrupar las variables y las restricciones, de tal forma que se ordenan los nodos por ganancias. Estos nodos se intercambiarán en cada pasada, de tal forma que se guardarán los mejores movimientos y los realiza. La función de coste será proporcional al número de nodos de entrada y salida de cada grupo, de tal forma que cuantos menos relaciones tengan los grupos (menor número de entradas y salidas), la solución será mejor. El primer algoritmo

para la migración de grupos fue el propuesto por Kernighan & Lin, el cual busca la partición en la que las interrelaciones entre los grupos es mínima. No obstante, existen otras variantes al algoritmo anterior en las que cambian la forma de producir los movimientos de operaciones entre los grupos.

Para aplicar dicho algoritmo, se eligen dos grupos como partición inicial. De esta partición se obtienen las ganancias de cada par de operaciones, entendiéndose como el número de relaciones externas al grupo menos el número de relaciones internas al grupo (desde cada operación) menos el doble de las relaciones entre las dos operaciones. La iteración se realiza intercambiando de grupos aquellas dos operaciones con una mayor ganancia (obviamente las ganancias únicamente hay que hallarlas entre operaciones de diferentes grupos), hasta que todas las ganancias toman un valor 0 o negativo. Las restricciones de las diferentes operaciones serán introducidas como ponderaciones en las ganancias.

Supongamos el ejemplo mostrado en la figura 3.7 se muestra un ejemplo de la utilización del algoritmo propuesto por Kernighan-Lin. La optimización se ha realizado en tres iteraciones (que ha dependido en gran medida de la partición inicial). En dicha figura se muestra las particiones, así como las ganancias de cada par de operaciones (ganancia de primera operación + ganancia de la segunda operación - 2 · interrelaciones entre las dos operaciones). Una vez que se ha detectado el primer cambio, es decir el cambio de las operaciones C y H, no hace falta considerar estas dos operaciones para las siguientes iteraciones (ya que se ha detectado su correcto lugar). No obstante, en el proceso completo se ha considerado todas las operaciones (aunque su lugar ya haya sido optimizado) para comprobar la aseveración anterior, es decir, la independencia de las reubicaciones realizadas con el agrupamiento final. También se muestra otro agrupamiento igualmente mínimo según este algoritmo, ya que todas las ganancias son iguales o menores que 0. No obstante, desde este último agrupamiento se puede observar un agrupamiento mínimo (con menos relaciones que los anteriores), el cual nunca se conseguiría con este método ya que el número de operaciones de cada grupo nunca cambia, luego se observa uno de los mayores inconvenientes de este método: no se garantiza un mínimo global.

Entre las principales ventajas que tiene esta técnica podemos encontrar las siguientes:

- Tiempos de ejecución bajos
- Escapa de algunos mínimos locales

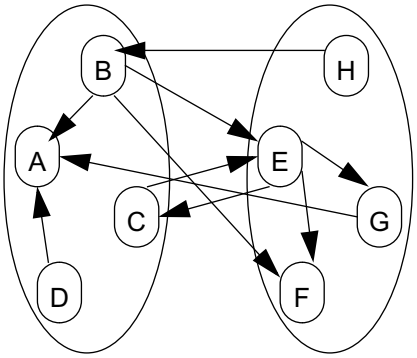
Mientras que muestra los siguientes inconvenientes:

- Muy dependiente de la solución inicial, es decir, es muy irregular
- No garantiza un mínimo global

Por lo tanto, una de sus principales aplicaciones es el refinamiento de otras técnicas.

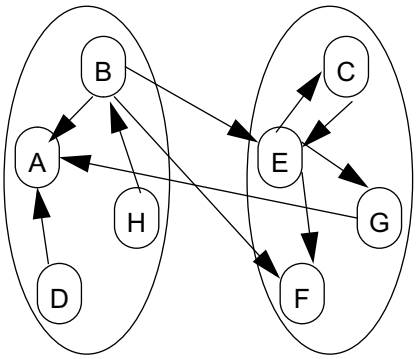
3.3.3. Clustering (Agrupamiento)

La técnica de clustering o agrupamiento consiste en ir agrupando cada una de las operaciones en función de su relación con las demás. Esta técnica presenta dos variantes en función del peso de cada operación: clustering uniforme, cuando todas las operaciones tienen el mismo peso; clustering basado en fuerzas, cuando las operaciones tienen diferente peso.



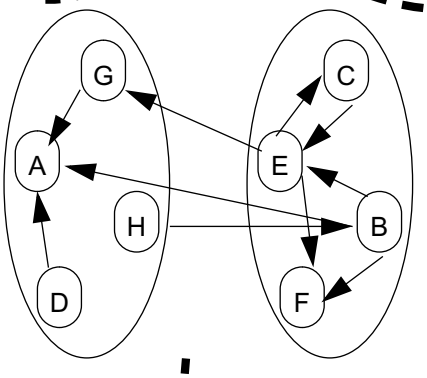
GAE = (1-2)A + (3-2)E - 2·(0) = 0 GCE = (2-0)C + (3-2)E - 2·(2) = -1
GAF = (1-2)A + (1-1)F - 2·(0) = -1 GCF = (2-0)C + (1-1)F - 2·(0) = +2
GAG = (1-2)A + (1-1)G - 2·(1) = -3 GCG = (2-0)C + (1-1)G - 2·(0) = +2
GAH= (1-2)A + (1-0)H - 2·(0) = 0 GCH= (2-0)C + (1-0)H - 2·(0) = +3
GBE = (3-1)B + (3-2)E - 2·(1) = +1 GDE = (0-1)D + (3-2)E - 2·(0) = 0
GBF = (3-1)B + (1-1)F - 2·(1) = 0 GDF = (0-1)D + (1-1)F - 2·(0) = -1
GBG = (3-1)B + (1-1)G - 2·(0) = +2 GDG = (0-1)D + (1-1)G - 2·(0) = -1
GBH = (3-1)B + (1-0)H - 2·(1) = +1 GDH = (0-1)D + (1-0)H - 2·(0) = 0

INTERCAMBIAR OPERACIONES C Y H



GAE = (1-2)A + (1-4)E - 2·(0) = -4 GHE = (0-1)H + (1-4)E - 2·(0) = -4
GAF = (1-2)A + (1-1)F - 2·(0) = -1 GHF = (0-1)H + (1-1)F - 2·(0) = -1
GAG = (1-2)A + (1-1)G - 2·(1) = -3 GHG = (0-1)H + (1-1)G - 2·(0) = -1
GAC= (1-2)A + (0-2)C - 2·(0) = -3 GHC= (0-1)H + (0-2)C - 2·(0) = -3
GBE = (2-2)B + (1-4)E - 2·(1) = -5 GDE = (0-1)D + (1-4)E - 2·(0) = -4
GBF = (2-2)B + (1-1)F - 2·(1) = -2 GDF = (0-1)D + (1-1)F - 2·(0) = -1
GBG = (2-2)B + (1-1)G - 2·(0) = 0 GDG = (0-1)D + (1-1)G - 2·(0) = -1
GBC = (2-2)B + (0-2)C - 2·(0) = -2 GDC = (0-1)D + (0-2)C - 2·(0) = -3

PARTICIÓN ÓPTIMA



GAE = (1-2)A + (1-4)E - 2·(0) = -4 GHE = (1-0)H + (1-4)E - 2·(0) = -2
GAF = (1-2)A + (0-2)F - 2·(0) = -3 GHF = (1-0)H + (0-2)F - 2·(0) =-1
GAB = (1-2)A + (2-2)B - 2·(0) = -1 GHB = (1-0)H + (2-2)B - 2·(1) =-1
GAC= (1-2)A + (0-2)C - 2·(0) = -3 GHC= (1-0)H + (0-2)C - 2·(0) = -1
GGE = (1-1)G + (1-4)E - 2·(1) = -5 GDE = (0-1)D + (1-4)E - 2·(0) =-4
GGF = (1-1)G + (0-2)F - 2·(0) = -2 GDF = (0-1)D + (0-2)F - 2·(0) = -3
GGB = (1-1)G + (2-2)B - 2·(0) = 0 GDB = (0-1)D + (2-2)B - 2·(0) = -1
GGC = (1-1)G + (0-2)C - 2·(0) = -2 GDC = (0-1)D + (0-2)C - 2·(0) = -3

PARTICIÓN ÓPTIMA

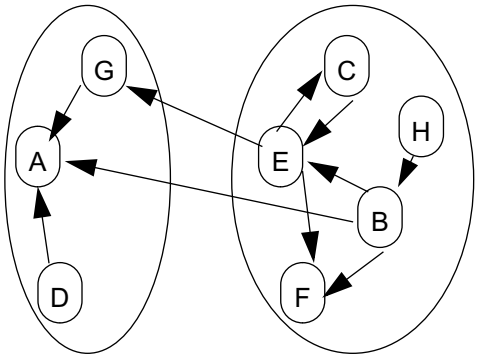


Figura 3.7.- Ejemplo del algoritmo de Kernighan-Lin.

El método uniforme comienza con una semilla (módulo con mayor número de conexiones). Seguidamente se colocan las operaciones muy conectadas con las ya colocadas de tal forma que colocamos todas las operaciones en una rejilla. Una vez que tengamos completada la rejilla, sólo nos queda agrupar las operaciones en dos grupos para su implantación vía hardware o software.

El método basado en fuerzas en un vector fuerza que depende de las relaciones con el resto de operaciones y la importancia de la operación. El método de colocación se obtiene realizando los siguientes pasos:

- Se colocan los dos o tres primeras operaciones utilizando el método uniforme.
- Cada nueva operación se coloca en un punto de la rejilla de tal forma que se minimiza la suma de fuerzas (estando el punto óptimo en la anulación de fuerzas)

Siguiendo con las operaciones del ejemplo de la figura 3.7, empezamos colocando la operación con mayor número de conexiones, es decir, la operación E. Seguidamente colocaremos la siguiente operación conectada a la anterior y con un mayor número de conexiones, obtenido las conexiones de las operaciones G, B, F y C. Procederemos del mismo modo hasta colocar todas las operaciones. Por último, sólo nos queda agrupar las diferentes operaciones de tal forma que las relaciones entre los grupos sea mínima. Con este método obtenemos una forma más manejable de representar gráficamente las operaciones antes de la agrupación, y por lo tanto, consiguiendo que los grupos sean gráficamente inconexos. Este algoritmo se puede observar en la figura 3.8.

3.3.4. Otras técnicas

Además de las técnicas anteriores, también podemos encontrar referencias a otras técnicas de particionado como pueden ser la programación lineal y los algoritmos genéticos.

La solución utilizada en programación lineal se basa en la formulación matemática de un problema de optimización combinatoria, reduciéndose a obtener un conjunto de inecuaciones entre las variables del sistema, los costes y las restricciones. Estas inecuaciones serán de primer orden (dependencia lineal) con respecto a las variables.

La utilización de algoritmos genéticos es una técnica relativamente reciente basada en el proceso de evolución de las especies. En esta técnica se dispondrá de una población con todas las soluciones posibles (codificadas), y con el objetivo final de obtener la solución más óptima. Para seleccionar una solución u otra se utilizan básicamente tres reglas:

- Sobrevive el mejor
- Cambio de solución
  - Cruces entre soluciones codificadas
  - Mutaciones de una solución codificada

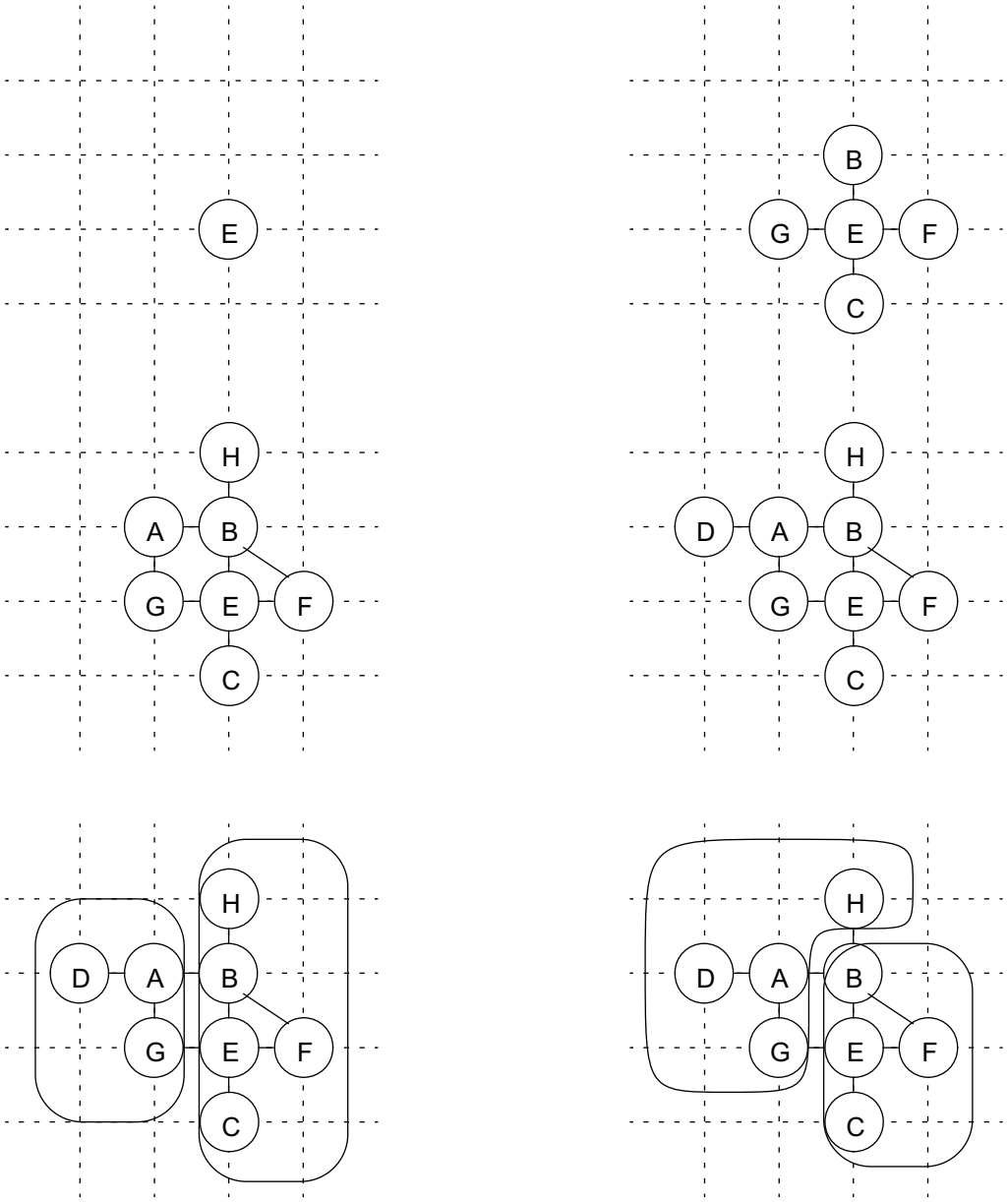


Figura 3.8.- Ejemplo del algoritmo de clustering