TEMA 6. INTRODUCCIÓN A LAS ARQUITECTURAS CON PARALELISMO EXPLÍCITO

6.1 INTRODUCCIÓN A LAS ARQUITECTURAS PARALELAS

Hasta ahora se ha estudiado principalmente el procesamiento a nivel del procesador. La segmentación es en cierto modo un mecanismo de paralelismo (varias instrucciones consecutivas son ejecutadas de forma solapada, casi en paralelo). Los procesadores superescalares también realizan procesamiento paralelo (dos o más instrucciones se lanzan al mismo tiempo por existir varios cauces de instrucciones).

Sin embargo, todos esos sistemas están basados en la arquitectura Von Neumann (propuesta en el año 1946 – se caracteriza principalmente por su versatilidad y sencillez), con un procesador y la memoria principal donde se guardan los datos y las instrucciones. Esta arquitectura ha ido perfeccionándose aplicando paralelismo a la unidad de control, a la unidad operativa o al sistema de acceso a la memoria principal; pero, siguen teniendo un único flujo de instrucciones.

Se puede decir que no hay una frontera clara entre la arquitectura monoprocesador y la arquitectura masivamente paralela (las actuales arquitecturas monoprocesador son máquinas paralelas a nivel de instrucción). Las arquitecturas monoprocesador han ido evolucionando para poder hacer frente a las fuertes demandas de computación existentes.

Al final de los años 80 surgen una amplia gama de máquinas que buscan en el paralelismo una forma de superar las limitaciones inherentes a la arquitectura SERIE (arquitectura Von Neumann – máquina versátil y sencilla). El paralelismo viene justificado por dos causas principales:

- La máxima potencia de cálculo alcanzable con arquitectura serie (la velocidad de transmisión en el silicio es de $3 \cdot 10^7$ m/s, para una pastilla de 3 cm de lado, el retardo longitudinal de las señales es de 10^{-9} s; por lo tanto, para un funcionamiento serie – una única operación en cada momento – la máxima potencia de cálculo alcanzable será de 10^9 operaciones de coma flotante por segundo (1 GFLOP)).
- El coste. La fabricación de grandes series de circuitos VLSI tienen un coste reducido.

El inconveniente principal que van a presentar las máquinas paralelas es su gran dificultad a la hora de realizar su programación. Para estos casos se aconsejan más los lenguajes de tipo *funcional* (a menudo denominados procedimentales – lenguaje que se programa mediante funciones, devuelve un nuevo estado de resultado y recibe como entrada el resultado de otras funciones) (en estos, se describe qué se quiere obtener y se deja que el compilador y el sistema operativo determinen cómo se van a realizar los cálculos) que los de tipo *imperativo* (lenguaje que se programa mediante una serie de comandos, agrupados en bloques y compuestos de órdenes condicionales que permiten al programa ir a otros bloques de comandos si se cumple la condición). Los lenguajes de tipo funcional permiten de forma natural la explotación del paralelismo de los procesos.

Se puede interpretar un computador vectorial como el primer paso hacia la paralelización. Un procesador vectorial incluye instrucciones vectoriales dentro del repertorio de instrucciones del mismo y segmenta las operaciones que hay que realizar sobre las distintas componentes que definen los vectores sobre los que actúan dichas instrucciones vectoriales. Esto hace que un computador vectorial sea básicamente una extensión del concepto de segmentación.

6.2 CLASIFICACIÓN DE LOS COMPUTADORES

Se pueden establecer varias clasificaciones de los computadores según se atienda a distintos conceptos:

SEGÚN LA ORGANIZACIÓN DEL PROGRAMA:

- **Ordenadores de flujo de control** → Usan información explícita de los flujos de control para producir la ejecución de las instrucciones.

  Mecanismo de datos empleado[1] → Por referencia (referencias incluidas en las instrucciones que están siendo ejecutadas, para acceder a los contenidos de la memoria principal)
  Mecanismo de control empleado[2] → Secuencial (pasa un único hilo de control de instrucción en instrucción)

- **Ordenadores de flujo de datos** → Utilizan la disponibilidad de los operandos para disparar la ejecución de las operaciones.

  Mecanismo de datos empleado → Por valor (que genera un argumento en tiempo de ejecución que se repite y se da a cada instrucción que accede para almacenarlo como un valor)
  Mecanismo de control empleado → Paralelo (los soportan los tokens de datos que transportan datos desde instrucciones consumidoras y contribuyen a la activación de las instrucciones consumidoras)

Tanto en unos como en otros se construyen sus programas a partir de instrucciones primitivas de tamaño fijo con programas de más alto nivel construidos desde secuencias de estas instrucciones primitivas y operaciones de control.

- **Ordenadores de reducción** → Usan la necesidad que tiene un resultado de disparar la operación que generará el resultado requerido. Presentan dos formas básicas de organización:

  - **Reducción de cadenas** → Emplean los siguientes mecanismos de datos y de control:

    Mecanismo de datos empleado → Por valor (ventaja cuando manipula instrucciones simples)
    Mecanismo de control empleado → Recursivo

[1] El mecanismo de datos define la manera en que un argumento particular es usado por las instrucciones.
[2] El mecanismo de control define cómo una instrucción produce la ejecución de otra u otras instrucciones y el modelo de control resultante.

- **Reducción de grafos** → Con los siguientes mecanismos de datos y control:

  Mecanismo de datos empleado → Por referencia (ventaja cuando están implicadas estructuras grandes)
  Mecanismo de control empleado → Recursivo

Para los ordenadores de reducción se construyen los programas a partir de estructuras de programas de alto nivel sin la necesidad de operadores de control.

SEGÚN LA ORGANIZACIÓN DE LA MÁQUINA:

- **Centralizada** → Consta de un procesador único, comunicaciones y memoria. Una instrucción activa única pasa la ejecución a una instrucción sucesora específica. Por ejemplo, la máquina tradicional de Von Neumann (de flujo de control).
- **Comunicación por paquetes** → Se usa un cauce de ejecución de instrucciones circular en cada procesador; las comunicaciones y memorias están enlazadas por asociaciones de trabajo. Por ejemplo, la NEC μPD 7281 (de flujo de datos).
- **Manipulación de expresiones** → Usa recursos idénticos en una estructura regular; cada recurso contiene un procesador, comunicaciones y memoria. El programa consiste en una gran estructura con partes activas y otras suspendidas temporalmente. Por ejemplo, una estructura regular de transputers T414.

SEGÚN EL TIPO DE ARQUITECTURA:

Otra clasificación general de las arquitecturas de computador establece los siguientes tipos:

- **Arquitectura Von Neumann** → Tiene las siguientes características fundamentales:

  - Un procesador, que tiene un único elemento de cálculo, comunicaciones y memoria.
  - Una organización lineal de celdas de memoria de tamaño fijo.
  - Un lenguaje máquina de bajo nivel con instrucciones que ejecutan operaciones simples sobre operandos elementales.
  - Control de cálculo centralizado y secuencial.

- **Arquitectura Harvard** → Se trata simplemente de una pequeña mejora de la arquitectura Von Neumann, en la que existen memoria de datos y memoria de instrucciones separadas y con caminos de comunicación separados. De esta forma se evita en parte el cuello de botella del bus de la arquitectura Von Neumann; permitiendo además acelerar la ejecución de las instrucciones. En la Figura 6.1 se muestra esta arquitectura básica.
- **Arquitecturas Paralelas** → Con paralelismo tanto a nivel interno (paralelismo implícito) como externo (paralelismo explícito) al procesador.

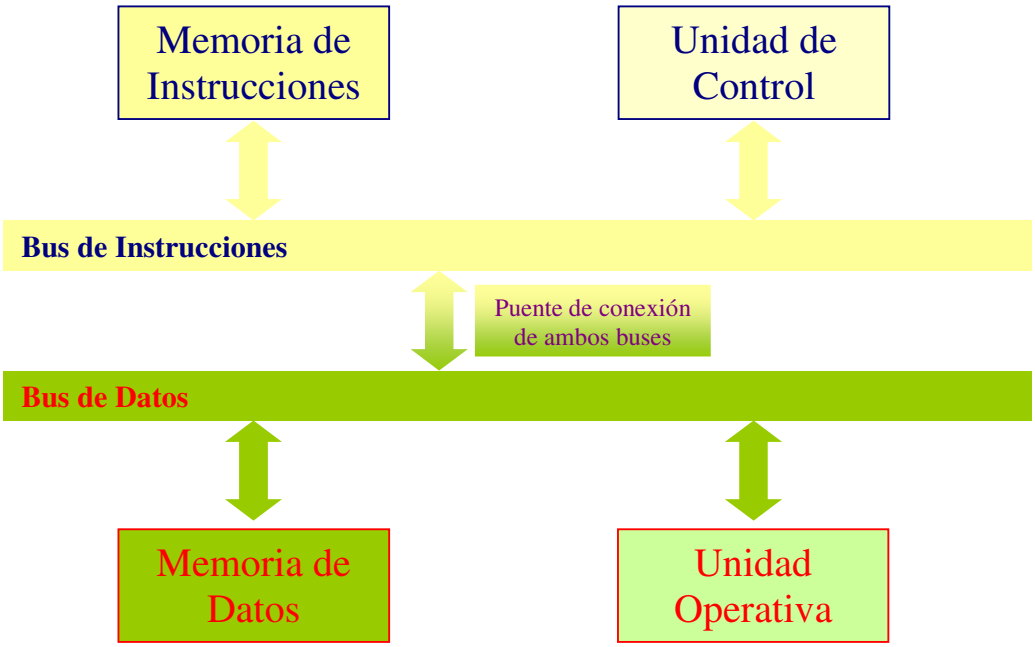


Figura 6.1 Esquema básico de la arquitectura Harvard.

CLASIFICACIÓN DE FLYNN:

La clasificación más popular de los sistemas computadores probablemente sea la propuesta por Michael J. Flynn en el año 1972. Esta taxonomía de las arquitecturas está basada en el flujo de datos y el flujo de instrucciones en un sistema computador. El flujo de instrucciones como conjunto de instrucciones secuenciales que son ejecutadas por un único procesador; y, el flujo de datos como flujo secuencial de datos requeridos por el flujo de instrucciones. De esta forma, Flynn clasifica los computadores en las siguientes cuatro categorías:

- **SISD (Single Instruction stream, Single Data stream – Flujo único de instrucciones y flujo único de datos)** → Se corresponde con el concepto de arquitectura serie de Von Neumann.
- **MISD (Multiple Instruction stream, Single Data stream – Flujo múltiple de instrucciones y flujo único de datos)** → Varias instrucciones actúan sobre el mismo y único flujo de datos. Este tipo de máquina se puede interpretar de dos formas diferentes:
    - Considerar la clase de máquinas que requerirían que unidades de procesamiento diferentes recibieran instrucciones distintas operando sobre los mismos datos → Algunos arquitectos de computadores dicen que es una arquitectura impracticable o imposible (actualmente no existen ejemplos de este tipo).
    - Considerarla como una clase de máquina donde un mismo flujo de datos fluye a través de numerosas unidades de procesamiento. Pueden ser encuadradas en este tipo, las arquitecturas altamente segmentadas del tipo:

- **Procesadores sistólicos** y **procesadores de frente de onda** → El procesador sistólico es un array regular de elementos de procesamiento, cada uno comunicándose con su vecino más cercano y operando sincrónicamente bajo el control de un reloj común, con una velocidad limitada por el procesador más lento en el array (en este caso hay varios contadores de programa, uno por cada procesador). El procesador de frente de onda consiste en un array regular de elementos de procesamiento, cada uno comunicándose con sus vecinos más cercanos, pero operando sin reloj global. Es concurrente y conducido por los datos. En este caso, el funcionamiento de cada procesador se controla localmente y se activa con la llegada de datos después de que su salida previa haya sido entregada al procesador vecino apropiado. El procesamiento avanza a través del array al ser pasados los datos por los procesadores a sus vecinos (en este caso también hay varios contadores de programa, uno por cada procesador). El array de procesadores del tipo frente de onda tiende a ser más eficiente que el de tipo sistólico cuando los tiempos de procesamiento son variables.
- **Procesadores vectoriales** → En el caso de que se consideren los elementos de un vector como pertenecientes al mismo dato y que las etapas del cauce representan múltiples instrucciones que son aplicadas sobre ese vector.

- **SIMD (Single Instruction stream, Multiple Data stream – Flujo de instrucción simple y flujo de datos múltiple)** → Una única instrucción es aplicada sobre diferentes datos al mismo tiempo. En estas máquinas, varias unidades de procesamiento diferentes son invocadas por una única unidad de control. Al igual que las MISD, las SIMD soportan procesamiento vectorial (matricial), asignando cada elemento del vector a una unidad funcional diferente para procesamiento concurrente. En este tipo de computadores se encajan los **computadores matriciales**.
- **MIMD (Multiple Instruction stream, Multiple Data stream – Flujo de instrucciones múltiple y flujo de datos múltiple)** → Constituyen los sistemas multiprocesadores propiamente dichos. Su estructura la constituyen varios procesadores serie, cada uno con una unidad de procesamiento (que soporta su flujo de datos) y una unidad de control (que soporta su flujo de instrucciones). Las máquinas MIMD son las máquinas más complejas pero son las que potencialmente ofrecen una mayor eficiencia en la ejecución concurrente o paralela; aquí la concurrencia implica que no sólo hay varios procesadores operando simultáneamente, sino que además hay varios programas (procesos) ejecutándose también al mismo tiempo. Tanto la estructura general como el nivel de interacción entre los distintos procesadores, permiten múltiples soluciones. Una de las características fundamentales de los multiprocesadores es el tipo de acoplo entre los procesadores que lo componen; diferenciándose las tres alternativas siguientes:

    - **Acoplo débil** → **S**e trata en realidad de varios computadores serie convencionales, que se pueden comunicar entre si a velocidades

relativamente altas (por ejemplo con red local de 10 MB/s). Idealmente se dispone de un único sistema operativo que realiza el reparto de carga. Su funcionamiento es eficiente siempre que las interacciones entre procesos de distintos procesadores sea pequeña o nula.

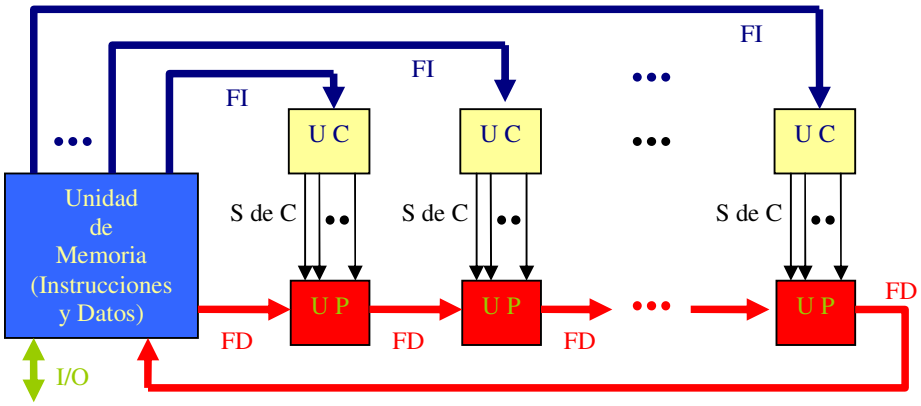
- **Acoplo moderado** → En este caso debe existir un sistema operativo único, así como un mapa de memoria único. Esta única memoria sirve de elemento de comunicación entre los distintos procesos pero, el acceso de los distintos procesadores a los distintos recursos del sistema no es homogéneo.
- **Acoplo fuerte** → Todos los procesadores del sistema pueden utilizar todos los recursos. Generalmente existe una memoria principal común, que se emplea también para la comunicación entre los distintos procesadores.

En la Figura 6.2 se muestran graficamente los esquemas planteados en la clasificación de Flynn. Puede observarse que un elemento fundamental de las arquitecturas paralelas es su *red de interconexión*, que permite que las diferfentes unidades del sistema computador se comuniquen entre si.

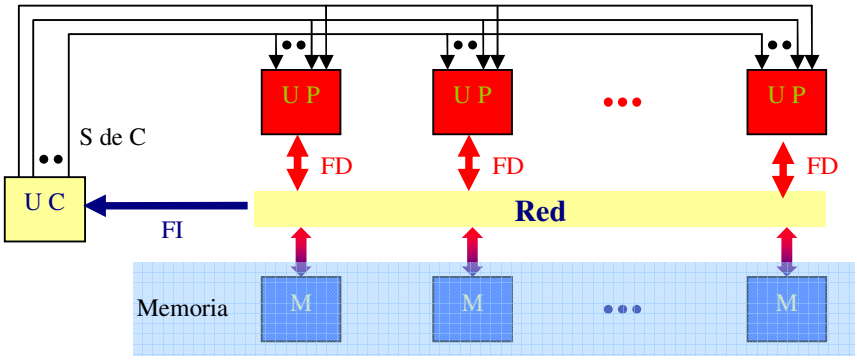
TIPOS DE PARALELISMO:

- **Paralelismo interno (**a la Unidad Central de Proceso**)** → Queda oculto a la arquitectura del computador. Es un paralelismo a nivel de estructura y/o construcción del computador, que se utiliza para aumentar su velocidad, pero sin modificar su funcionamiento básico. Aplicado a la arquitectura Von Neumann, permite construir computadores más rápidos pero, a nivel de ejecución de instrucciones máquina, es decir, a nivel de usuario, siguen siendo serie. En este apartado se encuentran las siguientes soluciones clásicas:
  - **Segmentación o pipe-line** → Paralelismo/solapamiento dentro de la Unidad de Control.
  - **División funcional** → Paralelismo/solapamiento dentro de la Unidad Operativa.
- **Paralelismo explícito o externo (**a la Unidad Central de Proceso**)** → Queda visible al usuario y, como consecuencia, éste debe encargarse de explotarlo de forma óptima.
  - **Flujo de control**:
    - **MISD**
    - **SIMD**
    - **MIMD:**
      - **Acoplo fuerte**
      - **Acoplo moderado**
      - **Acoplo débil**
  - **Flujo de datos**
  - **De reducción**

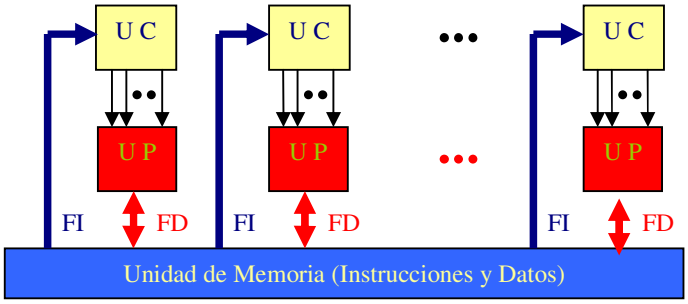
a) **SISD** (monoprocesador)



b) **MISD** (array sistólico y de frente de onda)



c) **SIMD** (procesador matricial)



d) **MIMD** (multiprocesador)

Figura 6.2 Clasificación de Flynn de las arquitecturas de computador.

6.3 CLASIFICACIÓN DE LOS SISTEMAS PARALELOS

La clasificación planteada por Flynn ha demostrado funcionar bastante bien para la tipificación de sistemas, y se ha venido usando desde décadas por la mayoría de los arquitectos de computadores. Sin embargo, los avances en tecnología y diferentes topologías han llevado a sistemas que no son tan fáciles de clasificar en la taxonomía establecida por Flynn. Por ejemplo, ni los procesadores vectoriales ni las arquitecturas híbridas encajan bien en esta clasificación.

Se propone otra clasificación (apuntes de *Arquitecturas Avanzadas* de Fernando Pardo Carpio; Universidad de Valencia) en la Figura 6.3 que no es más que una taxonomía ampliada de la de Flynn.

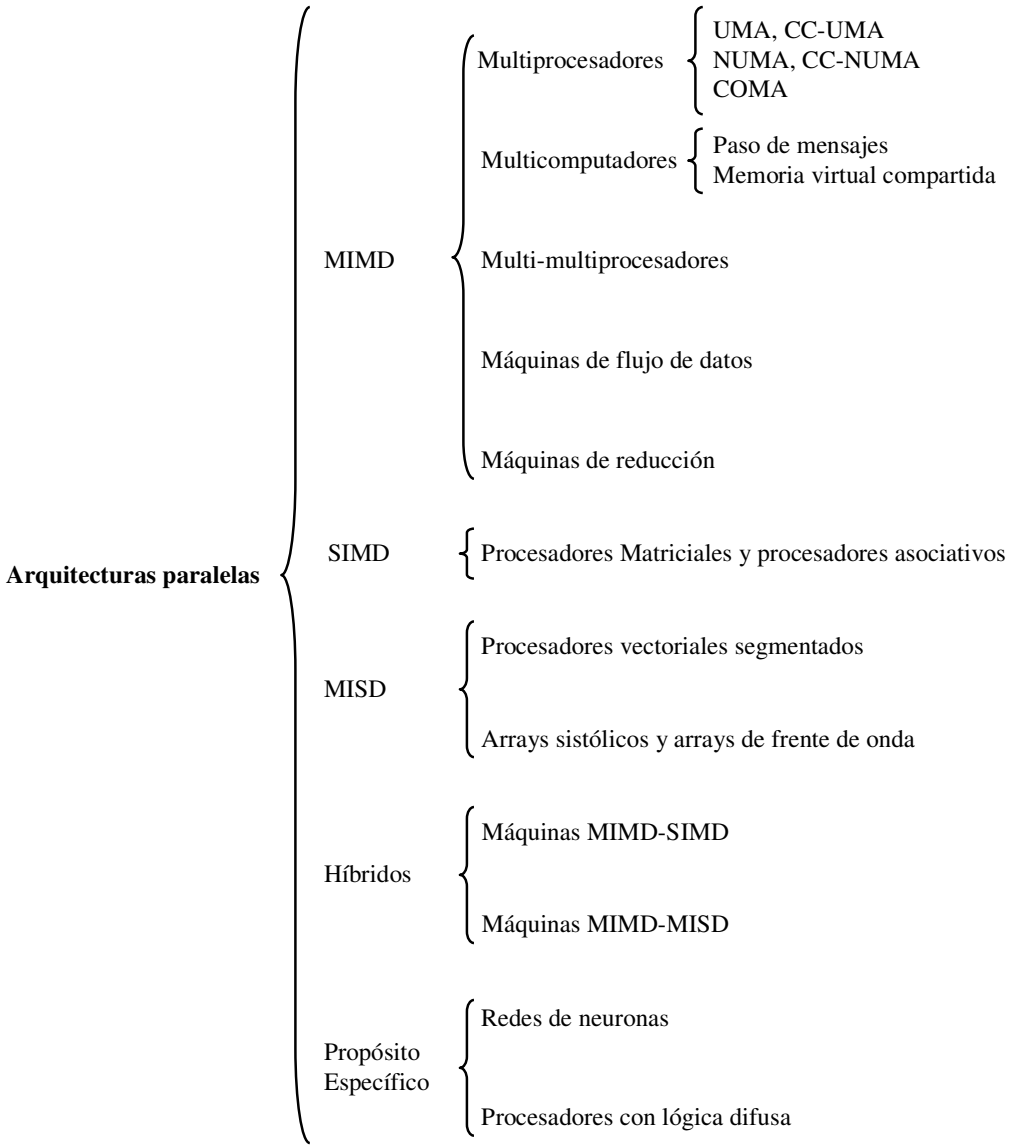


Figura 6.3 Clasificación de las arquitecturas paralelas.

MULTIPROCESADORES

Un *multiprocesador* puede entenderse como un computador paralelo compuesto por varios procesadores interconectados, que pueden compartir un mismo sistema de memoria. Los procesadores pueden configurarse para que cada uno ejecute una parte de un programa o varios programas al mismo tiempo. Generalmente un multiprocesador está formado por *n procesadores* (P_1, P_2, …, P_n) y *m módulos de memoria* (M_1, M_2, …, M_m). Una *red de interconexión* conecta cada procesador a un subconjunto de los módulos de memoria.

Por compartir los multiprocesadores los distintos módulos de memoria, pudiendo acceder a un mismo módulo de memoria varios procesadores, también se les llama *sistemas de memoria compartida*. Dependiendo de la forma en que los procesadores compartan la memoria, se tiene la siguiente subdivisión de los multiprocesadores:

- **Multiprocesador UMA** (*Uniform Memory Access* – Acceso a memoria uniforme) → La memoria física está uniformemente compartida por todos los procesadores (todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de memoria). Cada procesador puede tener su caché privada, y los periféricos también se comparten (esta compartición de los periféricos puede ser diferente según el sistema). Estos sistemas se catalogan como fuertemente acoplados por su alto grado de compartición de los recursos. La red de interconexión toma la forma de bus común, conmutador cruzado o una red multietapa, como se verá posteriormente.

  Cuando todos los procesadores tienen el mismo acceso a todos los periféricos, el sistema se llama *multiprocesador simétrico*. En este caso, todos los procesadores tienen la misma capacidad para ejecutar programas, tal como el Kernel o las rutinas de servicio de E/S. En un *multiprocesador asimétrico*, sólo un subconjunto de los procesadores pueden ejecutar programas; a los que pueden o al que puede (muchas veces es sólo uno), se les llama *maestros*, y al resto de procesadores se les llama *attached processors* - procesadores adheridos. En la Figura 6.4 se muestra el diagrama de bloques correspondiente a los *multiprocesadores UMA*.

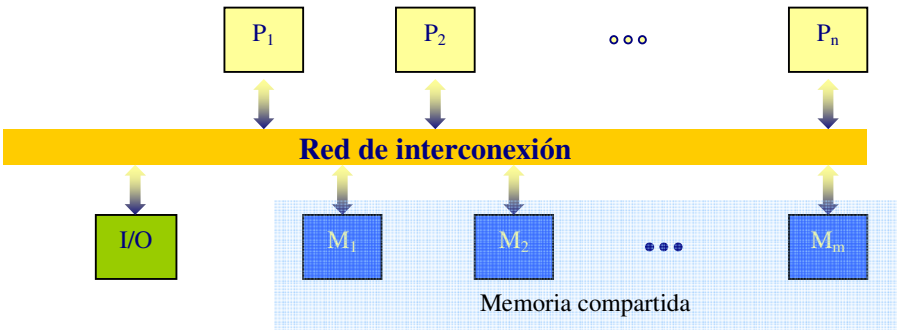


Figura 6.4 Multiprocesador UMA.

También, es frecuente encontrar arquitecturas de acceso uniforme que además tienen coherencia de caché, a estos sistemas se les suele llamar **CC-UMA** (*Cache Coherent Uniform Memory Access* – Acceso a memoria uniforme con coherencia de caché).

- **Multiprocesadores NUMA** (*Non Uniform Memory Access* – Acceso a memoria no uniforme) → Es un sistema de memoria compartida donde el tiempo de acceso depende del lugar donde se encuentre localizado el acceso. La Figura 6.5 muestra una posible configuración de multiprocesador NUMA, donde hay memoria local para cada procesador pero toda es compartida. Es decir, la memoria ha sido distribuida físicamente entre los procesadores, reduciendo el tiempo de acceso a la memoria local (LM_i) e incrementando la escalabilidad; a estos computadores paralelos también se les denomina multiprocesadores DSM (*Distributed Shared Memory* – Memoria compartida distribuida). En cierto modo han seguido algunas tendencias establecidas previamente por los *multicomputadores* pero, en este caso, los accesos a las memorias remotas se hacen en respuesta a accesos a la memoria en vez de en llamadas al sistema como ocurre en los *multicomputadores*. Otras posibles configuraciones incluyen los sistemas basados en agrupaciones (*clusters*) de sistemas como el de la misma Figura 6.5, que se comunican a través de otra red de comunicación que pueda incluir una memoria compartida global.

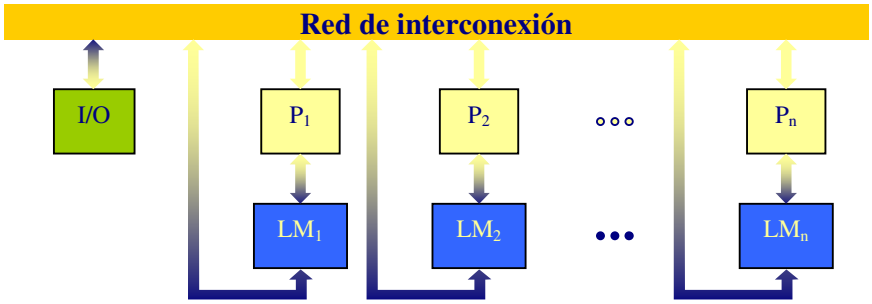


Figura 6.5 Multiprocesador NUMA.

La ventaja principal en estos sistemas es que el acceso a la memoria local es más rápido que en los UMA; aunque un acceso a memoria no local es más lento. Se intenta que la memoria utilizada por los procesos que ejecuta cada procesador, se encuentre en la memoria de dicho procesador para que los accesos sean lo más locales posible.

Se puede añadir al sistema, además de las memorias locales, una memoria de acceso global. En este caso se dan tres posibles patrones de acceso para un procesador P_i. El más rápido es el acceso a su memoria local (LM_i); le sigue el acceso a memoria global (GM); y, el más lento, es el acceso a la memoria local del resto de procesadores (LM_j, $j \neq i$).

Al igual que hay sistemas CC-UMA, también existe el modelo de acceso a memoria no uniforme con coherencia de caché **CC-NUMA** (*Cache Coherent Non Uniform Memory Access* – Acceso a memoria no uniforme con coherencia de caché); consiste en memoria compartida distribuida y directorio de caché.

- **Multiprocesadores COMA** (*Cache Only Memory Access* – Acceso a memoria únicamente por caché) → Es un multiprocesador que sólo usa caché como memoria (Figura 6.6). En realidad se trata de un caso especial del modelo NUMA, las memorias distribuidas se han convertido en cachés. No hay jerarquía de memoria en cada módulo procesador: todas las cachés forman un mismo espacio global de direcciones; por lo tanto, parte de la memoria caché de un procesador puede tener datos que no han sido demandados por el procesador asociado. El acceso a las cachés remotas se realiza a través de los directorios distribuidos de las cachés. Dependiendo de la red de interconexión empleada, se pueden utilizar jerarquías en los directorios para ayudar en la localización de copias de bloques de caché. El emplazamiento inicial de datos no es crítico puesto que el dato acabará estando en el lugar en que se use más.

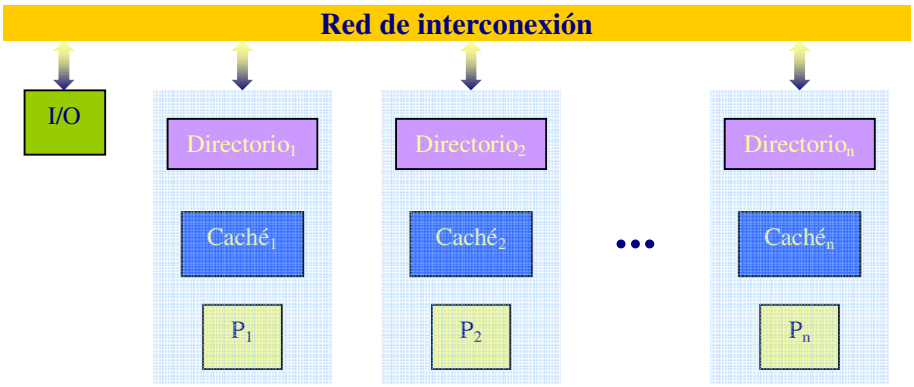


Figura 6.6 Multiprocesador COMA.

Dado que puede ser costoso que toda la memoria sea caché, Hagwersen, Landin y Haridi (en 1992) describen un sistema COMA con una porción de memoria que actúa como caché real, siendo el resto de la memoria lo que ellos denominan una *memoria de atracción* (los datos son atraidos a la memoria de atracción más cercana al procesador que lo demandó).

- **Multicomputadores (paso de mensajes)** → Un multicomputador se puede ver como un computador paralelo en el que cada procesador tiene su propia memoria local. La memoria del sistema se encuentra distribuida entre los distintos procesadores de cada computador, pudiendo cada procesador direccionar su memoria local (acceso directo) exclusivamente; el acceso a la memoria del resto de procesadores (acceso indirecto) debe hacerlo por *paso de mensajes*. Este acceso local y privado a la memoria es lo que diferencia los multicomputadores de los multiprocesadores.

  El diagrama de bloques de un multicomputador (Figura 6.7) es prácticamente igual al visto en la Figura 6.5 para el multiprocesador NUMA, la diferencia viene dada porque la red de interconexión no permite un acceso directo entre memorias, sino que la comunicación se realiza por paso de mensajes.

  La transferencia de datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia entre procesadores se realiza por tanto por múltiples transferencias entre procesadores dependiendo de cómo esté establecida la red.

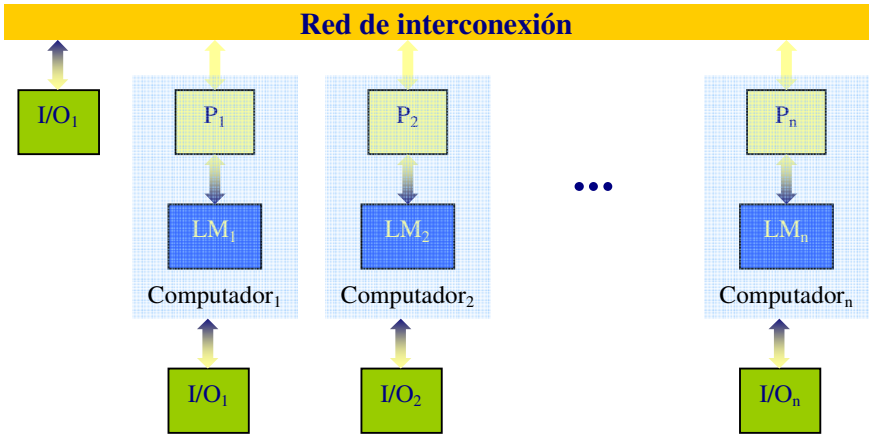


Figura 6.7 Multicomputador.

Dado que la memoria está distribuida entre los diferentes elementos de proceso, a estos sistemas se les llama también *distribuidos*. Pero no hay que confundirlos con los multiprocesadores NUMA, que también tienen la memoria distribuida pero, para estos multiprocesadores, la memoria distribuida también es compartida; cosa que no ocurre en los multicomputadores. Además, dado que en los multicomputadores se explota mucho la localidad, a estos sistemas se les llama débilmente acoplados (los módulos funcionan de forma casi independiente unos de otros).

La programación de multicomputadores no es una tarea sencilla. El programador en este caso debe encargarse de:

1. La distribución de código y datos entre los procesadores de manera eficiente.
2. Realizar las llamadas que se encarguen de enviar los datos que necesiten otros procesadores.

- **Multicomputadores con memoria virtual compartida** → En un multicomputador, un proceso de usuario puede construir un espacio global de direccionamiento virtual. El acceso a dicho espacio global de direcciones virtuales se puede realizar por software mediante un *paso de mensajes explícito*. En las bibliotecas de paso de mensajes hay siempre rutinas que permiten a los procesos aceptar mensajes de otros procesos. Una lectura se realiza mediante el envío de una petición al proceso que contiene el objeto. La petición por medio del paso de mensajes puede quedar oculta al usuario por poder ser generada por el compilador que tradujo el código de acceso a una variable compartida. De esta manera el usuario se encuentra programando un sistema aparentemente basado en memoria compartida, cuando en realidad se trata de un *sistema basado en el paso de mensajes*. A este tipo de sistemas, en los que se realiza un *paso de mensajes implícito*, se les denomina *multicomputadores con memoria virtual compartida*.

  Otra forma de tener un espacio de memoria virtual compartido consiste en el *uso de páginas*. En este tipo de sistemas, una colección de procesos tienen una

región de direcciones compartida pero, sólo las páginas que son locales son accesibles de forma directa para cada proceso. Si se produce un acceso a una página remota, entonces se genera un fallo de página y, el sistema operativo inicia una secuencia de pasos de mensaje para transferir la página y ponerla en el espacio de direcciones del proceso.

- **Multi-multiprocesadores** → Básicamente consisten en sistemas multicomputadores donde cada uno de sus elementos (computador) es a su vez un sistema multiprocesador (UMA, NUMA, COMA). La comunicación y compartición de datos entre los multiprocesadores se realiza por la técnica de paso de mensajes (explícitamente o implícitamente).

- **Máquinas de flujo de datos** → Hay dos formas de procesar la información, una es mediante la ejecución en serie de una lista de comandos y la otra es la ejecución de un comando demandado por los datos disponibles. La primera forma empezó con la máquina de Von Neumann y continuó con sucesivas modificaciones que la convirtieron en actuales multiprocesadores que permiten paralelismo. La segunda forma de ver el procesamiento de datos en principio puede parecer menos directa pero, desde el punto de vista de la paralelización resulta mucho más interesante ya que las instrucciones se ejecutan en el momento en que están disponibles los datos para ellas y, por supuesto, se deben poder ejecutar todas las instrucciones demandadas en un instante determinado (debe existir un hardware que lo permita). Hay algunos lenguajes que se adaptan a este tipo de arquitecturas como son Prolog, ADA, etc., es decir, lenguajes que exploten de una u otra manera la concurrencia de instrucciones.

  Como se ha dicho, en una arquitectura de flujo de datos, una instrucción está lista para su ejecución cuando los datos que necesita están disponibles. La disponibilidad de los datos se consigue por la canalización de los resultados de las instrucciones ejecutadas con anterioridad hacia los operandos de las instrucciones que esperan. Esta canalización forma un flujo de datos que van disparando las instrucciones a ejecutar; por esto se evita la ejecución de instrucciones basada en *contador de programa*.

  Las instrucciones en un computador de flujo de datos son *puramente autocontenidas*; es decir, no direccionan variables en una memoria compartida global, sino que llevan los valores de las variables en ellas mismas. En una máquina de este tipo, la ejecución de una instrucción no afecta a otras que estén listas para su ejecución. De esta manera, varias instrucciones pueden ser ejecutadas simultáneamente; llevando a un alto grado de concurrencia y paralelización.

  La Figura 6.8 muestra el diagrama de bloques de una máquina de flujo de datos. Las instrucciones, junto con sus operandos, se encuentran almacenados en la *memoria de datos e instrucciones* (I/D M). Cuando una instrucción está lista para ser ejecutada, se envía a uno de los *elementos de proceso* (EP) a través de la *red de arbitraje*. Cada elemento de proceso (EP) es un procesador simple con memoria local limitada. El elemento de proceso (EP), después de procesar la instrucción, envía el resultado a su destino a través de la *red de distribución*.

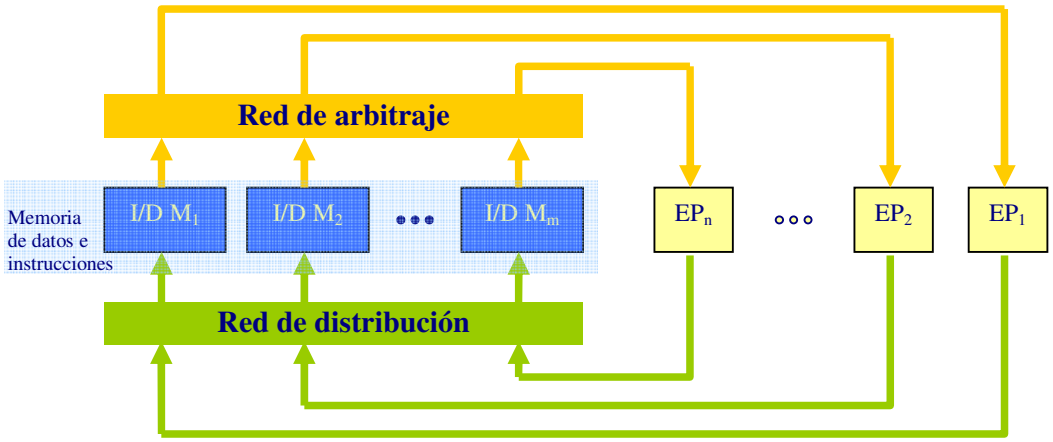


Figura 6.8 Diagrama de bloques de una máquina de flujo de datos. Máquina de Dennis.

- **Máquinas de reducción** → Como ya se ha dicho anteriormente, las máquinas de reducción usan la necesidad que tiene un resultado de disparar la operación que generará el resultado requerido. La estructura de este tipo de máquinas (Figura 6.9) responde al mismo esquema que corresponde a las máquinas MIMD en general; en este caso, la red de conexión tiene por objeto conseguir balancear la carga de trabajo entre los distintos procesadores que integran el conjunto.

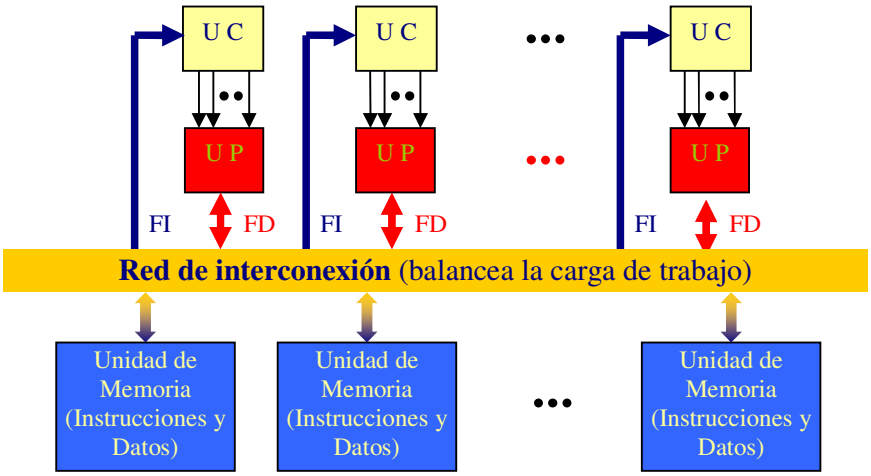


Figura 6.9 Esquema de la máquina de reducción.

- **Procesadores matriciales y procesadores asociativos** → Los procesadores matriciales son la arquitectura representativa del tipo SIMD, es decir, hay una sóla instrucción que opera concurrentemente sobre múltiples datos. Las instrucciones típicas de la máquina SIMD son como las instrucciones vectoriales de una máquina vectorial, que operan sobre varios elementos de datos en una sóla instrucción, ejecutándose de manera segmentada en una única unidad funcional. De forma distinta en una máquina SIMD, una única instrucción invoca a varias unidades funcionales a la vez (por ejemplo, una verdadera SIMD

podría tener 64 flujos de datos, simultáneamente, hacia 64 ALU para formar 64 sumas en el mismo ciclo de reloj.

La idea de utilización de los procesadores matriciales es *explotar el paralelismo en los datos* de un problema más que paralelizar la secuencia de ejecución de las instrucciones. El problema se paraleliza dividiendo los datos en particiones sobre las que se pueden realizar las mismas operaciones. Un tipo de datos altamente particionable es el formado por vectores y matrices; por eso a estos procesadores se les llama *matriciales*.

Un procesador matricial consiste en un conjunto de *elementos de proceso* y un *procesador escalar* que operan bajo una *unidad de control*. La unidad de control busca y decodifica las instrucciones de la memoria central y las manda bien al procesador escalar o bien a los nodos procesadores dependiendo del tipo de instrucción. La instrucción que ejecutan los nodos procesadores es la misma y se ejeuta de forma simultánea; los datos serán los de cada memoria de procesador y por tanto serán diferentes. Por todo esto, un procesador matricial requiere un único programa para controlar todas las unidades de proceso (reducido tamaño de la memoria de programa comparado con los MIMD).

Igual que en las máquinas vectoriales, los computadores SIMD tienen una mezcla de las instrucciones de los SISD y las propias de los SIMD. Se puede decir que hay un computador SISD para realizar operaciones como bifurcaciones o cálculos de direcciones, que no necesitan paralelismo.; y otro computador para las instrucciones SIMD, que se difunden a todas las unidades de ejecución, cada una de las cuales tiene su propio conjunto de registros. También, como en las máquinas vectoriales, determinadas unidades de ejecución pueden ser inhabilitadas durante una instrucción SIMD. De forma distinta a las máquinas vectoriales, las máquinas SIMD masivamente paralelas (más de 100 elementos de proceso) cuentan con redes de interconexión o de comunicación para intercambiar datos entre los elementos de procesamiento.

Las máquinas SIMD son apropiadas en situaciones semejantes para las que son adecuadas las instrucciones vectoriales (tratamiento de arrays en *bucles for*). Y, tienen su mayor debilidad en las *sentencias case*, donde cada unidad de ejecución debe realizar una operación diferente sobre su dato dependiendo del dato (las unidades de ejecución con el dato erróneo son inhabilitadas para que las unidades con los datos adecuados puedan continuar.

El término *computador matricial* se usa exclusivamente para computadores SIMD que utilizan memoria de acceso aleatorio convencional; y el término *computador asociativo* para computadores SIMD que utilizan memoria asociativa.

Se puede establecer una clasificación de los computadores SIMD, atendiendo al número de unidades de control que utilizan, al tipo de memoria empleada (asociativa y no asociativa) y a la realización del proceso en sección de bits o en sección de palabras:

- *Procesador matricial en sección de palabras*

- *Procesador matricial en sección de bits*
- *Procesadores asociativos en sección de palabra*
- *Procesadores asociativos en sección de bits*
- *Procesadores SIMD-Múltiple* (*MSIMD*)

En la Tabla 6.1 se muestran distintos computadores SIMD comerciales.

Computador	**Arquitectura**	**Referencias**
Unger	Matric. Sec. Pal.	Propuesto por Unger (1958)
Solomon	Matric. Sec. Pal.	Propuesto por Slotnick (1962)
VAMP	Matric. Sec. Pal.	Propuesto por Senzig y Smith (1965)
ILLIAC	Matric. Sec. Pal.	ILLIAC-IV (1972)
BSP	Matric. Sec. Pal.	Desarrollado por Burroughs y suspendido en 1979
CLIP	Matric. Sec. Bit	Desarrollado en University College, Londres (1976)
DAP	Matric. Sec. Bit	Desarrollado por ICL, Inglaterra (1981)
MPP	Matric. Sec. Bit	Desarrollado por Goodyear Aerospace
PEPE	Asoc. Sec. Pal.	Desarrollado por Burroughs Corp. and System, Dev Corp.
STARAN	Asoc. Sec. Bit	Desarrollado por Goodyear Aerospace Corp.
OMEN	Asoc. Sec. Bit	Desarrollado por Sanders Associates (1976)
RELACS	Asoc. Sec. Bit	Propuesto para máquina de base de datos (1979)
MAP	MSIMD Sec. Pal.	Propuesta por Nutt (1977)
PM	MSIMD Sec. Pal.	Propuesta por Briggs y Hwang y cols. (1979)
Phoenix	MSIMD Sec. Pal.	Propuesta por Feierbach y Stevenson (1979)
NASF	Matric. Sec. Pal.	Propuesta en Stevens (1979)

Tabla 6.1 Sistemas de computadores SIMD comerciales.

La mayoría de los procesadores asociativos han sido en sección de bits, aunque también los hay en sección de palabras (PEPE). La mayoría de los procesadores asociativos se diseñaron para realizar recuperaciones rápidas de información y operaciones de bases de datos.

Resulta evidente que los computadores SIMD son sistemas de propósito especial; para un entorno de problemas específicos pueden alcanzar un rendimiento elevado, sin embargo, cuando se habla de aplicaciones de propósito general, la programación y vectorización son difíciles de resolver.

Los computadores MSIMD constituyen una subclase especial de computadores MIMD. En un procesador matricial múltiple existen mútiples flujos de instrucciones; cada flujo de instrucciones maneja múltiples conjuntos de datos, igual que hace una matriz SIMD. Los computadores MSIMD ofrecen una flexibilidad de aplicación mayor que una máquina SIMD única.

Atendiendo a la organización del mapa de memoria, los computadores SIMD se clasifican según los siguientes tipos:

- De memoria distribuida (Figura 6.10) → Cada elemento de proceso (EP) dispone de su bloque de memoria. En este caso, los operandos se distribuyen antes de la ejecución.
- De memoria compartida (Figura 6.11) → Los elementos de memoria pueden ser compartidos por todas los elementos de proceso (EP). El mapa de memoria se compone de un número *m* de módulos en paralelo; para evitar en lo posible los conflictos por accesos simultáneos al mismo

módulo se procura que el número *n* de elementos de proceso (EP) sea distinto de *m* (normalmente primos entre si). La red de interconexión también debe favorecer el acceso libre de conflictos.

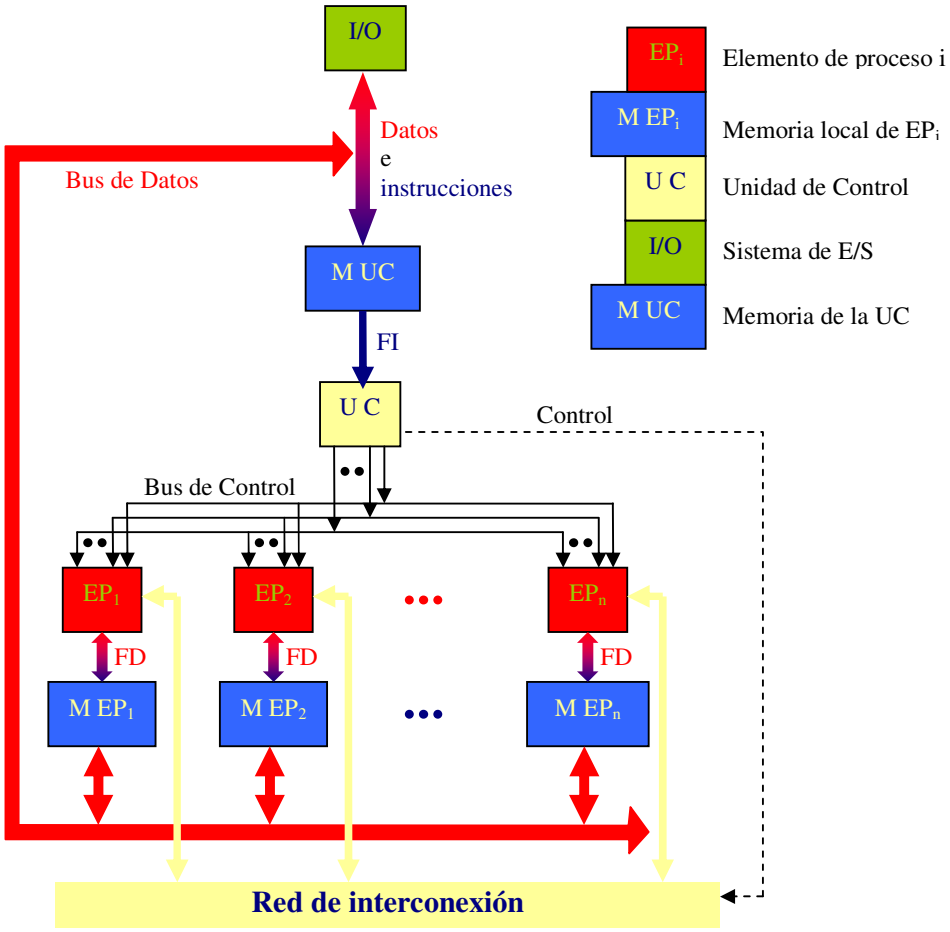


Figura 6.10 Máquina SIMD de memoria distribuida.

La máquina SIMD de memoria distribuida está configurada con *n* elementos de proceso (EP) sincronizados, y todos ellos bajo el control de una única unidad de control (UC). Cada elemento de proceso (EP) es esencialmente una unidad aritmético-lógica asociada con registros de trabajo y lleva asociado cada uno de ellos una memoria local (M EP) para el almacenamiento de los datos distribuidos. La unidad de control (UC) también tiene su memoria principal para el almacenamiento de los programas (M UC). Los programas del sistema y los programas de usuario se ejecutan bajo control de la unidad de control (UC). Los programas de usuario se cargan en la memoria de la unidad de control (M UC) desde una fuente externa. La función de la unidad de control es decodificar todas las instrucciones y determinar dónde deberían ejecutarse las instrucciones decodificadas. Las instrucciones escalares o de tipo de control se ejecutan directamente dentro de la unidad de control. Las instrucciones vectoriales se transmiten a los elementos de proceso (EP) para ejecutarlas en modo distribuido y alcanzar así un paralelismo espacial gracias a la multiplicidad de unidades aritméticas (elementos de proceso (EP)).

Todos los elementos de proceso (EP) realizan la misma función sincronizadamente en modo pasobloqueo bajo el mandato de la unidad de control (UC). Los operandos vectoriales se distribuyen por las memorias locales (M EP_i) antes de la colección paralela en el conjunto de elementos de proceso (EP_i). Los datos distribuidos pueden cargarse en las memorias locales (M EP_i) desde una fuente externa a través del bus de datos del sistema o a través de la unidad de control (UC) en modo difusión (*broadcast*) usando el bus de control. Se utilizan esquemas de enmascaramiento para controlar el estado de cada elemento de proceso (EP_i) durante la ejecución de una instrucción vectorial. Cada EP_i puede estar activo o inactivo durante un ciclo de instrucción. Se utiliza un vector de enmascaramiento para controlar el estado de todos los EP_i. Los intercambios de datos entre EP_i se efectúan a través de una red de comunicación entre EP_i, que realiza todas las funciones necesarias de encaminamiento y manipulación de datos. Esta red de interconexión está bajo control de la unidad de control (UC).

Un procesador matricial está conectado normalmente a un computador principal (host) a través de la unidad de control (UC). El computador principal es una máquina de propósito general que sirve como "director de explotación" del sistema entero, formado por el propio computador principal y la colección o matriz de procesadores. Las funciones del computador principal incluyen la administración de los recursos y la supervisión de los periféricos y la E/S en general. La unidad de control del procesador matricial (UC) supervisa directamente la ejecución de los programas, mientras que el computador principal realiza funciones de ejecutivo y de E/S con el mundo exterior. En este sentido, un procesador matricial puede considerarse también como un *computador especializado asociado*.

Otra forma de construir un computador matricial es la mostrada en la Figura 6.11, con memoria compartida. Esta configuración se diferencia de la anterior (con memoria distribuida) en dos aspectos:

1. Las memorias locales ligadas a los elementos de proceso se sustituyen ahora por módulos de memoria en paralelo, compartidos por todos los elementos de proceso mediante una red de alineamiento.
2. La red de interconexión de elementos de proceso se reemplaza por la red de alineamiento entre elementos de proceso y memoria, que sigue siendo controlada por la unidad de control (UC).

Es deseable que la red de alineamiento permita accesos libres de conflictos a las memorias compartidas por parte de tantos elementos de proceso (EP_i) como sea posible.

Los procesadores matriciales comenzaron a ser bien conocidos gracias al desarrollo hardware y software del sistema Illiac-IV. Desde entonces se han construido muchas máquinas SIMD para satisfacer diferentes aplicaciones de procesamiento paralelo (PEPE – *Parallel Element Processing Ensemble*, de Burroughs; Staran de Goodyear Aerospace que son dos procesadores matriciales asociativos) (BSP – Burroughs Scientific Processor; MPP – Massively Parallel

Processor de Goodyear Aerospace que se basan en el diseño de l Illiac-IV y son una ampliación y mejora de éste).

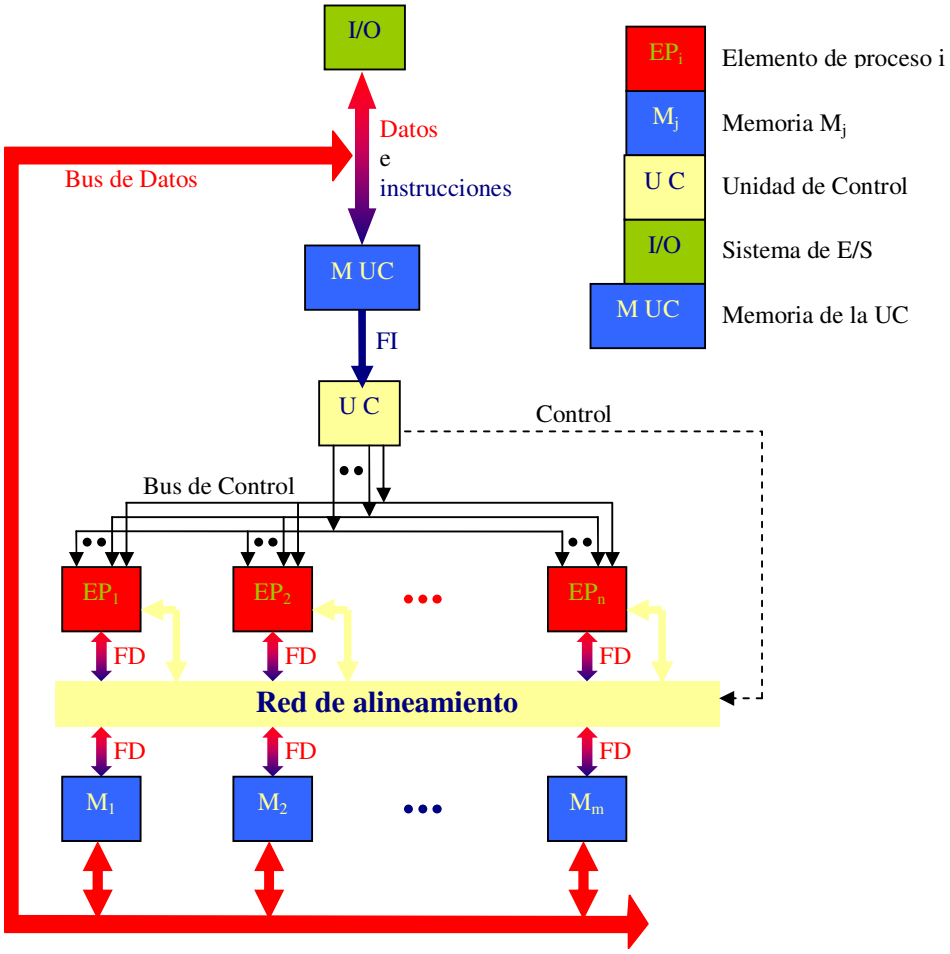


Figura 6.11 Máquina SIMD de memoria compartida.

Formalmente un computador SIMD se caracteriza por el siguiente conjunto de parámetros:

- **N** ≡ Número de elementos de proceso en el sistema
- **F** ≡ Conjunto de funciones de encaminamiento suministradas por la red de interconexión (de la Figura 6.10) o por la red de alineamiento (de la Figura 6.11).
- **I** ≡ El repertorio de instrucciones máquina para operaciones escalares, vectoriales, de encaminamiento de datos y de manipulación de la red.
- **M** ≡ Conjunto de esquemas de enmascaramiento, donde cada máscara particiona la colección de EP en dos subconjuntos disjuntos correspondientes a EP activos y EP inactivos.

$$SIMD = [N, F, I, M]$$

Ese modelo proporciona una base común para evaluar las diferentes máquinas SIMD.

El esquema de cada par EP_i y su memoria local asociada M EP_i se muestra en la Figura 6.12. El EP_i consta de un conjunto de registros de trabajo A_i, B_i y C_i, un registro de direcciones D_i (se emplea para contener los *m* bits de la dirección del EP_i), un registro índice local I_i (en una instrucción de referencia a memoria cada EP_i accede a su M EP_i con el desplazamiento indicado por su propio registro índice I_i; el registro índice I_i modifica la dirección de memoria global difundida desde la UC. De esta se pueden acceder a diferentes posiciones en diferentes M EP, simultáneamente con la misma dirección global especificada por la UC) , un registro de encaminamiento de datos R_i (este registro está conectado mediante la red de interconexión a otros R_j de otros elementos de proceso. Algunos procesadores matriciales usan dos registros encaminadores, uno para entrada y otro para salida), un indicador de estado S_i (para indicar qué elemento de proceso está activo y cual no), y un conjunto de operadores. En la unidad de control (UC) hay un registro índice global I y un registro de enmascaramiento M.

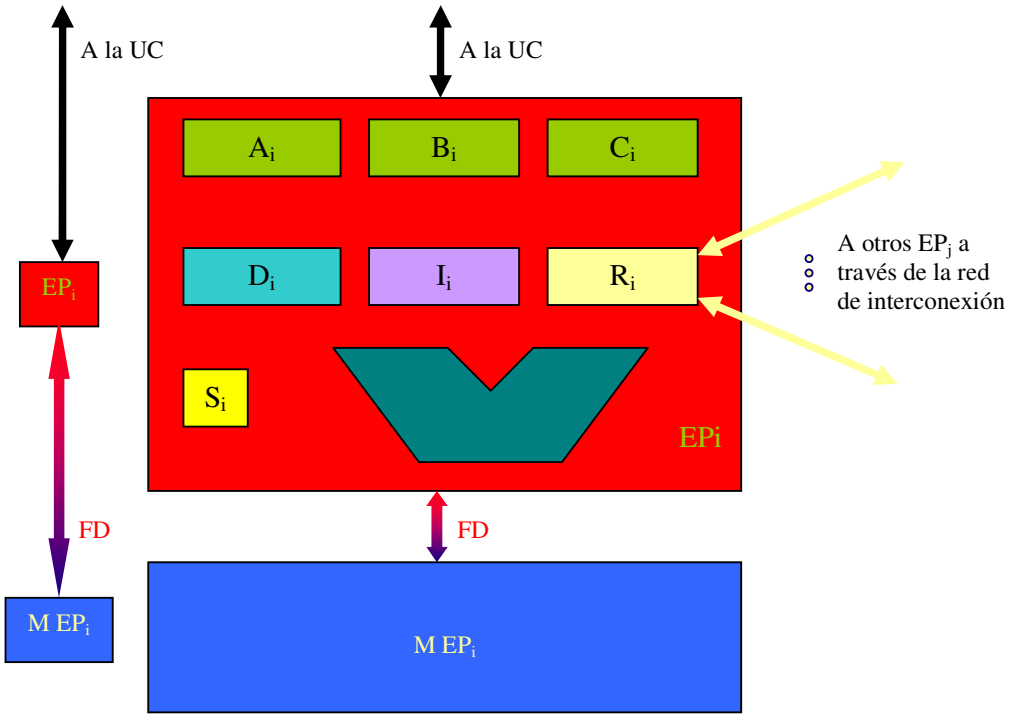


Figura 6.12 Componentes del par EP_i – M EP_i.

Los computadores SIMD-Múltiples (MSIMD) constituyen una subclase especial de computadores MIMD. En un procesador matricial múltiple existen múltiples flujos de instrucciones, y cada uno de estos flujo de instrucciones manejan múltiples flujos de datos (como en una matriz SIMD). Los computadores

MSIMD ofrecen una flexibilidad de aplicación mayor que una máquina SIMD única. En la Figura 6.13 se muestra el esquema representaivo de este tipo de sistemas.

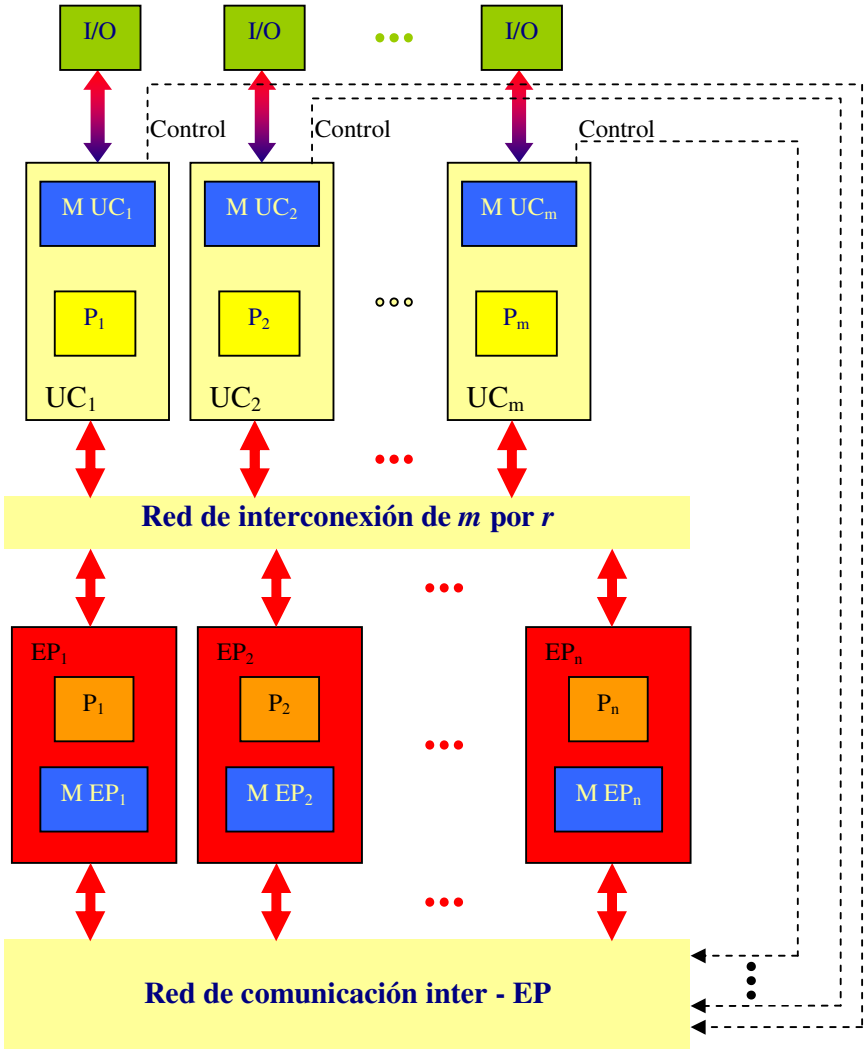


Figura 6.13 Máquina MSIMD (con SIMD de memoria distribuida).

Un procesador matricial de recursos compartidos (MSIMD) consta de dos o más UC_i que comparten un conjunto de EP_j asignables dinámicamente. El sistema opera con múltiples flujos de Simple-Instrucción Múltiples-Datos. Cada UC_i debe asignarse a un subconjunto de EP_j para la ejecución de un único trabajo vectorial (un proceso SIMD). Ejemplos de procesadores matriciales MSIMD son: el diseño original de Illiac IV con cuatro UC que comparten 246 EP, el Multi-Associative Processor (MAP) con ocho UC que comparten 1024 EP y el sistema PM propuesto en la Universidad de Purdue; este último es reconfigurable, pudiendo operar en modo MSIMD, en modo múltiple SISD o en modo MIMD.

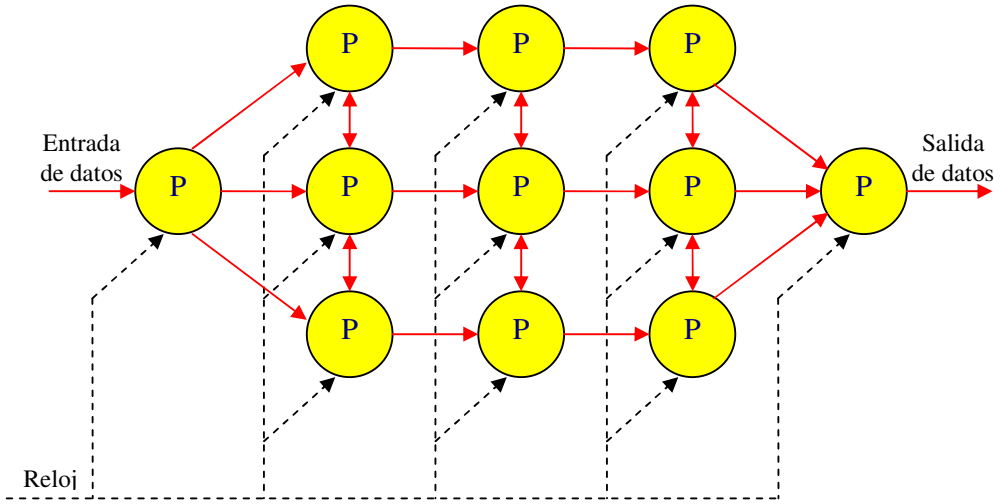
Algunas áreas de aplicación que se han sugerido para los procesadores matriciales son:

- Algebra matricial (multiplicación, descomposición e inversión).
- Cálculo de los autovalores de una matriz.
- Programación lineal y entera.
- Modelización meteorológica.
- Convolución.
- Filtrado y análisis de Fourier.
- Procesamiento de imágenes y reconocimiento de patrones.
- Experimentos en túnel de viento.
- Generación automática de mapas.
- Análisis de escenas en tiempo real.

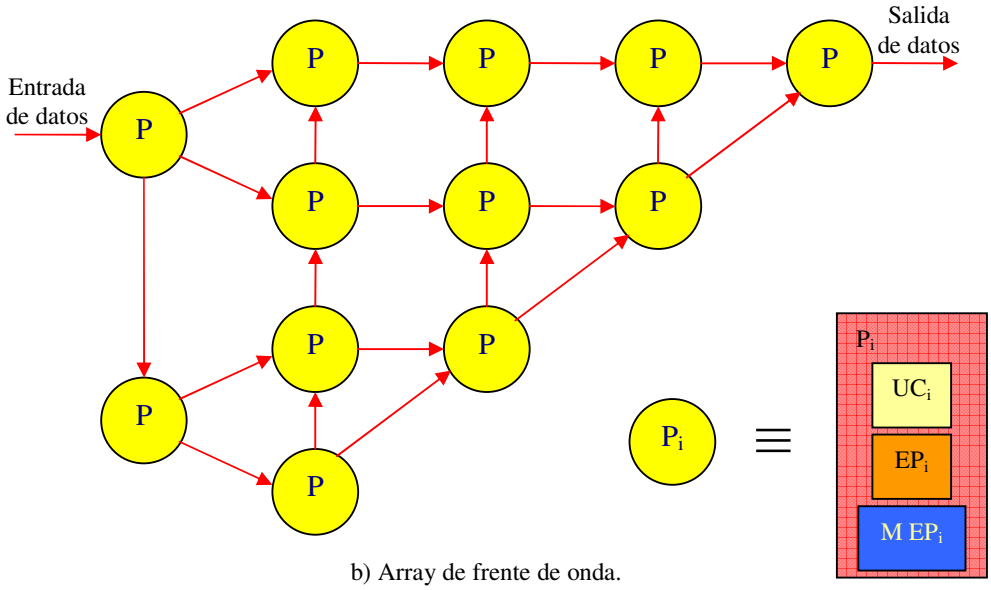
- **Procesadores vectoriales segmentados** → Un procesador vectorial ejecuta de forma segmentada instrucciones sobre vectores. La diferencia con los matriciales es que mientras los matriciales son comandados por las instrucciones, los vectoriales son comandados por flujos de datos continuos. Este tipo de computador se considera MISD puesto que varias instrucciones son ejecutadas sobre un mismo dato (el vector); consideración que, aunque es un poco confusa es ampliamente aceptada. Para profundizar más en el estudio de los procesadores vectoriales se puede acudir al tema 4 de la asignatura.
- **Arrays sistólicos y arrays de frente de onda** → Se trata de otro tipo de máquinas que se pueden considerar MISD. Tanto en un *array sistólico* como en un *array de frente de onda* hay un gran número de elementos de proceso (EP) idénticos, cada uno de ellos con memoria local (M EP). Estos elementos están colocados en forma de matriz (array) de manera que sólo están permitidas las conexiones con los elementos de proceso (EP) vecinos. Por lo tanto, todos los EP se encuentran organizados en una estructura segmentada de forma lineal o matricial. En el *array sistólico* (ver Figura 6.14 a), los datos fluyen de unos EP a sus vecinos a cada ciclo de reloj, y durante ese ciclo de reloj, o varios, los EP realizan una operación sencilla y de forma sincrónica; el adjetivo "sistólico" viene precisamente del hecho de que todos los procesadores vienen sincronizados por un único reloj que hace de "corazón", que hace moverse a la máquina. En el procesador de frente de onda (ver Figura 6.14 b), el funcionamiento de cada EP se controla localmente y se activa con la llegada de datos después de que su salida previa ha sido entregada al procesador vecino apropiado. El procesamiento avanza a través del array al pasar los datos los procesadores a sus vecinos. En este caso también tenemos, al igual que en el array sistólico, varios contadores de programa, uno por cada procesador. En los arrays sistólicos y de frente de onda, la comunicación con el mundo exterior únicamente se realiza a través de los *procesadores fronteras*.

- **Arquitecturas híbridas** → Se han visto dos formas de explotar el paralelismo: por un lado, la paralelización de código que se consigue con las máquinas de tipo MIMD; y, por otro lado, la paralelización de los datos conseguida con arquitecturas SIMD y MISD. En la práctica, el mayor beneficio en paralelismo viene de la paralelización de los datos; esto se debe a que el paralelismo de los datos explota el paralelismo proporcionalmente a la cantidad de datos que forman el cálculo a realizar. Sin embargo, muchas veces resulta imposible

explotar el paralelismo inherente en los datos del problema y se hace necesario utilizar tanto el paralelismo de control como el de datos. Por lo tanto, procesadores que tienen características de MIMD y SIMD (o MISD) al mismo tiempo, pueden resolver de forma efectiva un elevado rango de problemas.



a) Array sistólico.



b) Array de frente de onda.

Figura 6.14 Ejemplos de arrays sistólicos y arrays de frente de onda.

- **Arquitecturas específicas** → Estas arquitecturas son también conocidas muchas veces como *arquitecturas VLSI*, por implicar la mayoría de las veces la elaboración de circuitos específicos con una alta escala de integración.

  Un ejemplo de este tipo de arquitecturas son las *redes neuronales* (ANN – *Artificial Neural Network*). Consisten en un elevado número de elementos de

proceso muy simples que operan en paralelo. Estas arquitecturas se pueden utilizar para resolver aquellos problemas tales como el reconocimiento de patrones, comprensión del lenguajes etc., que le resultan fáciles a un humano y muy difíciles a una máquina. La diferencia principal con las arquitecturas clásicas es su forma de programación.; mientras en la arquitectura Von Neumann se aplica un programa o algoritmo para resolver un problema, una red de neuronas aprende a fuerza de aplicarle patrones de comportamiento.

La idea es la misma que en el cerebro humano: cada elemento de proceso es como una neurona con numerosas entradas provenientes de otros elementos de proceso y una única salida que va a otras neuronas o a la salida del sistema; dependiendo de los estímulos recibidos por las entradas a la neurona, la salida se activará o no dependiendo de una función de activación. Este esquema permite dos cosas: por un lado permite que la red realice una determinada función según el umbral de activación interno de cada neurona, y por otro, permite que pueda programarse la red mediante la técnica de ensayo-error.

Otro ejemplo de dispositivo de uso específico son los *procesadores basados en lógica difusa*. Estos procesadores tienen que ver con los principios del razonamiento aproximado. La lógica difusa intenta tratar con la complejidad de los procesos humanos eludiendo los inconvenientes asociados a la lógica clásica de dos valores.

6.4 PROCESAMIENTO MULTIHEBRA Y MULTIPROCESADORES MONOCHIPS

Una forma de medir las prestaciones de los procesadores es la velocidad a la que ejecuta instrucciones:

$$Velocidad\ (\text{MIPS}) = f\ (\text{MHz}) \times IPC\ (\text{instrucciones por ciclo})$$

Es por ello que los diseñadores han buscado la mejora de prestaciones abordando uno de los dos factores que intervienen en la velocidad de ejecución de instrucciones:

- Incrementando la frecuencia de reloj.
- Incrementando el número de instrucciones que se ejecutan en un ciclo.

El valor del IPC lo han aumentado los diseñadores utilizando un cauce segmentado primero y, después, utilizando varios cauces paralelos en las arquitecturas superescalares. Con los diseños segmentados, el principal problema es maximizar la utilización de cada etapa del cauce. Para ello, se han ideado mecanismos bastante complejos, tales como la ejecución de instrucciones fuera de orden. Sin embargo, esta aproximación tiene límites como consecuencia de los problemas relacionados con el aumento de la complejidad y el consumo de potencia.

Una alternativa que permite un paralelismo de instrucciones elevado sin incrementar ni la complejidad de los circuitos ni el consumo de potencia, es el ***procesamiento multihebra*** (*multithreading*). Básicamente, consiste en dividir la secuencia de instrucciones original en secuencias más pequeñas, denominadas ***hebras*** (*threads*), que pueden ejecutarse en paralelo.

Existe un número considerable de diseños mutihebra realizados, tanto en procesadores comerciales como experimentales. A continuación se realiza una breve revisión de los conceptos principales relacionados con el procesamiento multihebra.

PROCESAMIENTO MULTIHEBRA IMPLÍCITO Y EXPLÍCITO

El concepto de hebra utilizado para estudiar los procesadores multihebra puede ser o no ser el mismo que el concepto de hebra en un sistema operativo multiprogramado. Por lo tanto, resulta útil definir brevemente una serie de términos:

- **Proceso** → Un programa en ejecución en un computador. Un proceso reúne dos características clave:
  - *Propiedad de recursos***:** Un proceso dispone de un espacio de direcciones virtuales para almacenar la imagen del proceso, que consta del programa, los datos, la pila y demás atributos que definen el proceso. En determinadas ocasiones, un proceso puede tener el control o poseer recursos tales como memoria principal, canales de E/S, periféricos y ficheros.
  - *Planificación/ejecución***:** La ejecución de un proceso sigue un camino de ejecución (traza) a través de uno o más programas. Esta ejecución puede entremezclarse con la de otros procesos. Así, un proceso tiene un estado de ejecución (preparado, en ejecución, etc.) y una prioridad de asignación, y es la entidad que el sistema operativo se encarga de planificar y asignar.
- **Conmutación de proceso (**cambio de contexto – *context switching***)** (no confundir con conmutación de hebras) → Operación que cambia el proceso que se está ejecutando en el procesador por otro proceso. Para ello, almacena todos los datos de control, registros y demás información del primer proceso y los reemplaza con la información correspondiente al segundo.
- **Hebra** → Una unidad de trabajo dentro de un proceso que se puede asignar al procesador. Incluye un contexto de procesador (con el contador de programa y el puntero de pila) y su propia área de datos para la pila (para permitir las llamadas a subrutinas). Una hebra se ejecuta secuencialmente y puede interrumpirse para que el procesador pase a ejecutar otra hebra.
- **Conmutación de hebra** → el control del procesador pasa de una hebra a otra dentro de un mismo proceso. Usualmente, este tipo de conmutación es mucho menos costosa que la de proceso.

Así, mientras que las hebras aparecen involucradas en la planificación y ejecución, los procesos tienen que ver tanto con la planificación/ejecución como con la asignación de recursos. Las hebras de un mismo proceso comparten los mismos recursos; esta es la razón por la que un cambio de hebra consume mucho menos tiempo que la conmutación de procesos. Los sistemas operativos tradicionales, tales como las primeras versiones de Unix, no utilizaban hebras; la mayoría de los sistemas operativos actuales, tales como Linux, otras versiones de Unix y Windows, sí permiten el uso de hebras. Se puede distinguir entre ***hebras de nivel de usuario*** (*user-level threads*), visibles para los programas de aplicación, y ***hebras de nivel de núcleo*** (*kernel-level threads*), visibles sólo para el sistema operativo. Ambos tipos de hebras se pueden denominar ***hebras explícitas***, definidas en el software.

Hasta ahora, todos los procesadores comerciales y la mayoría de los experimentales han utilizado procesamiento multihebra explícito. Estos sistemas ejecutan concurrentemente instrucciones de hebras explícitas diferentes, bien entremezclando instrucciones de hebras diferentes en cauces compartidos o mediante ejecución paralela y cauces paralelos. El *procesamiento multihebra implícito* hace referencia a la ejecución concurrente de varias hebras extraídas de un único programa secuencial. Esta ***hebras implícitas*** pueden ser definidas estáticamente por el compilador o dinámicamente por el hardware.

APROXIMACIONES AL PROCESAMIENTO MULTIHEBRA EXPLÍCITO

Como mínimo, un procesador multihebra debe proporcionar un contador de programa distinto para cada una de las hebras que puedan ejecutarse concurrentemente. Los diseños cambian según la cantidad y el tipo de hardware que se añade para permitir la ejecución concurrente de las hebras. En general, se captan instrucciones para cada hebra; el procesador trata cada hebra separadamente y puede utilizar técnicas como la predicción de saltos, el renombramiento de registros u otras técnicas superescalares para optimizar la ejecución de una hebra. Lo que se aprovecha es un paralelismo entre hebras (*thread-level parallelism*) que, sumado al paralelismo entre instrucciones, puede proporcionar una mejora de prestaciones considerable.

En un sentido amplio, se pueden considerar cuatro aproximaciones principales del procesamiento multihebra:

- **Multihebra entrelazada (o procesamiento multihebra de grano fino)** → El procesador trabaja con dos o más contextos al mismo tiempo, conmutando entre uno y otro en cada ciclo de reloj. Si una hebra se bloquea debido a dependencias de datos o retardos de memoria, esa hebra se salta y se pasa a ejecutar una hebra que esté preparada.
- **Multihebra con bloqueo (o procesamiento multihebra de grano grueso)** → Las instrucciones de una hebra se ejecutan sucesivamente hasta que se produce un evento que puede ocasionar un retardo, tal como un fallo de caché. Este evento induce una conmutación a otra hebra. Esta aproximación es efectiva en un procesador con ejecución ordenada, en cuyo caso se produciría un atasco debido al retardo asociado a un evento como un fallo de caché.
- **Multihebra simultánea (SMT)** → Instrucciones correspondientes a hebras diferentes se emiten simultáneamente a las unidades funcionales de un procesador superescalar. Se combina así la capacidad de emisión de varias instrucciones del procesador superescalar y el uso de mútiples contextos correspondientes a las diferentes hebras.
- **Multiprocesador monochip** → En este caso existen varias copias del procesador en un solo circuito integrado y cada procesador actúa sobre hebras diferentes. La ventaja de esta aproximación es que el área del circuito integrado se utiliza eficientemente evitando la cada vez mayor complejidad de los nuevos diseños del cauce segmentado.

En las dos primeras aproximaciones, las instrucciones que pertenecen a hebras distintas no se ejecutan simultáneamente; en su lugar, el procesador es capaz de conmutar rápidamente de una hebra a otra, utilizando un conjunto de registros diferentes y otra

información de contexto, permitiendo un mejor aprovechamiento de los recursos de ejecución del procesador y evitando las elevadas penalizaciones asociadas a los fallos de caché u otros eventos con retardos asociados. La alternativa *SMT* supone una ejecución simultánea real de instrucciones de hebras diferentes, utilizando los recursos de ejecución repetidos. Los *multiprocesadores monochip* también permiten la ejecución simultánea de instrucciones de hebras diferentes.

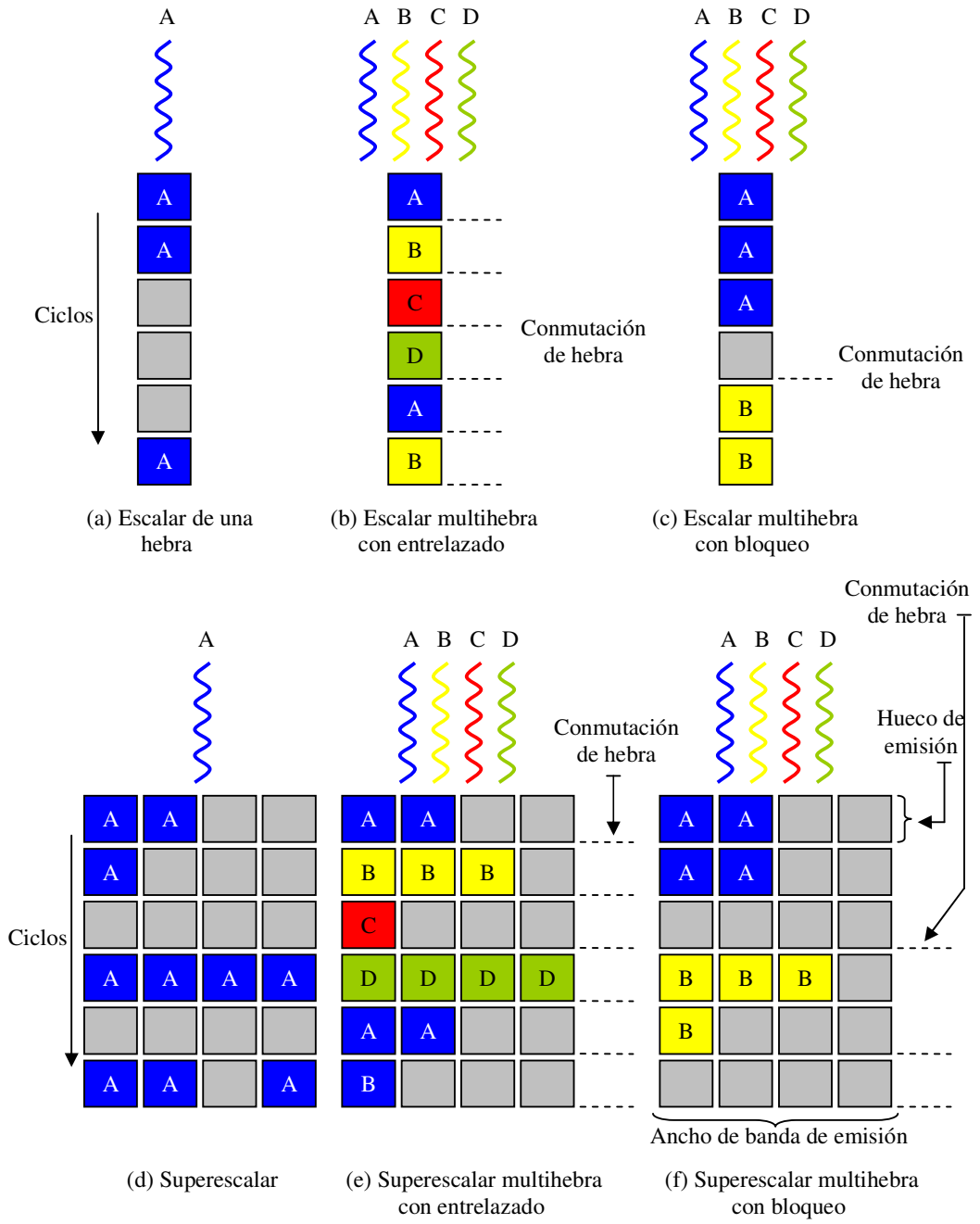


Figura 6.15 Alternativas para la ejecución de varias hebras.

Las Figura 6.15 y 6.16, muestran algunas de las posibles arquitecturas de cauce que implican procesamiento multihebra, y las compara con alternativas que no utilizan

procesamiento multihebra. Cada fila horizontal representa el hueco o huecos de emisión posibles por ciclo de ejecución; es decir, la anchura de cada fila corresponde al máximo número de instrucciones que pueden emitirse en un ciclo de reloj. La dimensión vertical representa la secuencia temporal de ciclos de reloj. Un hueco vacío (sombreado) significa un hueco de ejecución no utilizado en el cauce. Una no operación se indica con N.

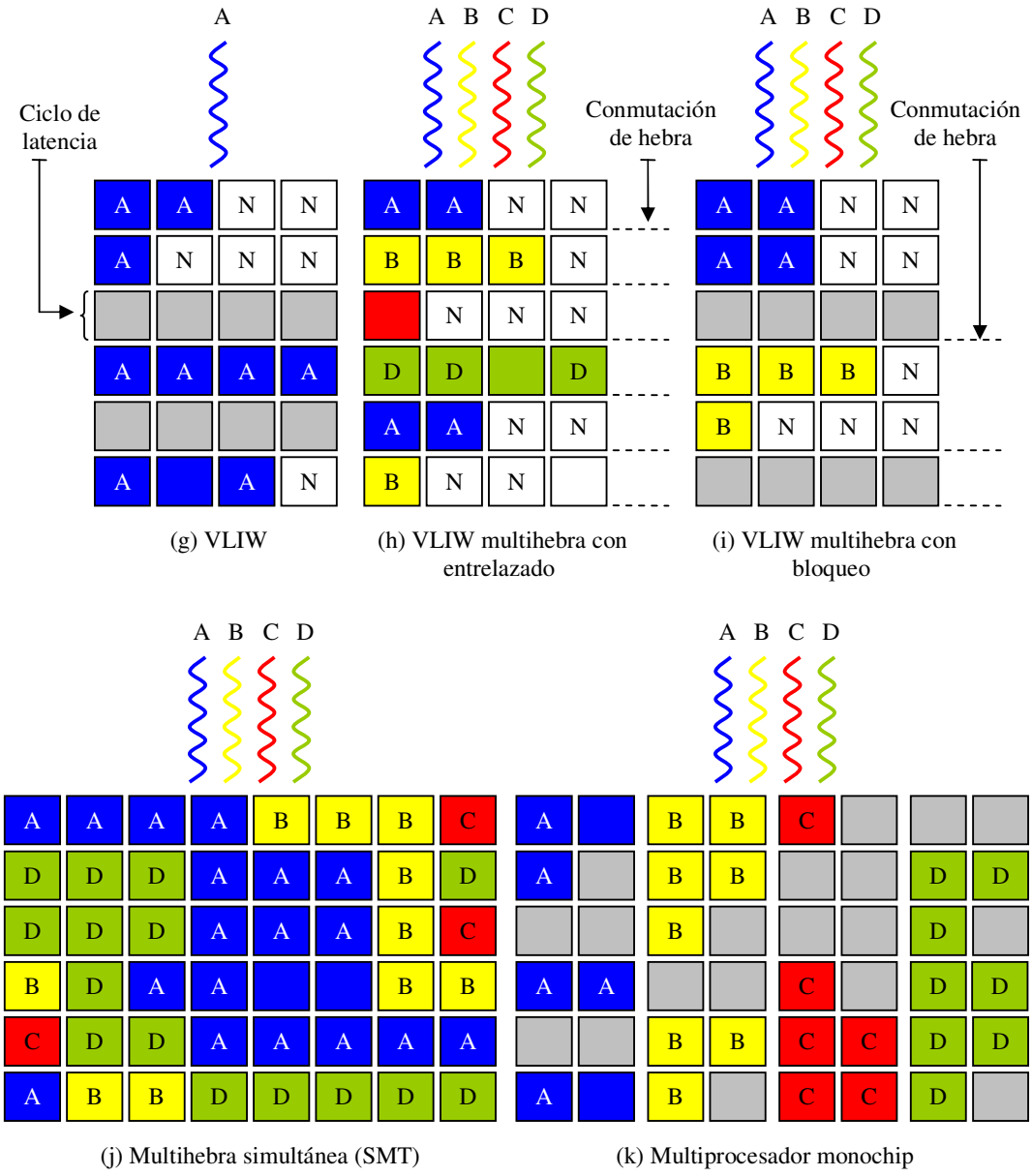


Figura 6.16 Alternativas para la ejecución de varias hebras.

Las primeras tres ilustraciones de la Figura 6.15 muestran distintas aproximaciones con un procesador escalar (con emisión de una única instrucción):

- **Escalar de una hebra** → Este tipo de cauce simple que se encuentra en las máquinas RISC y CISC tradicionales, sin procesamiento multihebra.

- **Escalar multihebra con entrelazado** → Esta es la aproximación al procesamiento multihebra más fácil de implementar. Al conmutar de una hebra a otra en cada ciclo, las etapas del cauce pueden mantenerse totalmente ocupadas, o casi totalmente ocupadas. El hardware debe ser capaz de conmutar entre contextos de hebras entre ciclos.
- **Escalar multihebra con bloqueo** → En este caso, cada hebra se ejecuta hasta que se produce un evento que ocasiona un retardo que podría detener el cauce, y que hace que el procesador conmute a otra hebra.

La figra 7.15 (c) muestra una situación en la que el tiempo necesario para realizar el cambio de hebra es igual a un ciclo, mientras que en la Figura 6.15 (b) la conmutación entre hebras no consume ningún ciclo de reloj. En este caso se asume que no hay control de dependencias de datos en el procesamiento multihebra entrelazado, con lo cual el diseño del cauce se simplifica y se podría conseguir la conmutación de hebras sin retardos. Sin embargo, según el diseño y la implementación concreta que se tenga. El procesamiento multihebra con bloqueo puede necesitar un ciclo de reloj para conmutar entre hebras, como se muestra en las figuras 6.15 y 6.16. Esto es así si una instrucción captada activa la conmutación de hebra y la hebra a la que pertenece dicha instrucción debe sacarse del cauce.

Aunque el procesamiento multihebra entrelazado parece proporcionar una mejor utilización del procesador que el procesamiento multihebra con bloqueo, esto se consigue sacrificando las prestaciones del procesamiento de cada hebra. Las distintas hebras compiten por la caché, dando lugar a un momento de la probabilidad de fallo de caché para cada una de las hebras.

Si el procesador puede emitir varias instrucciones por ciclo, aumentan las oportunidades de ejecución paralela. Las figuras 6.15 (d) a 6.15 (f) 6.16 (g) a 6.16 (i) muestran algunas diferencias entre procesadores que disponen de hardware para emitir cuatro instrucciones por ciclo. En todos estos casos, en cada ciclo únicamente se emiten instrucciones de una misma hebra. Las alternativas que se muestran son las siguientes:

- **Superescalar** → Es la aproximación superescalar básica sin procesamiento multihebra. Hasta hace relativamente poco, esta era la aproximación más potente para aprovechar el paralelismo dentro del procesador. Hay que tener en cuenta que, en algunos ciclos, no todos los huecos de emisión se utilizan. En esos ciclos, se emiten menos instrucciones que el número máximo permitido, lo que se denomina *pérdida horizontal*. Durante otros ciclos de instrucción no se utiliza ningún hueco de emisión. Esto significa que no se emite ninguna instrucción y se denomina *pérdida vertical*.
- **Superescalar multihebra con entrelazado** → En cada ciclo, se emiten tantas instrucciones como se pueda de una única hebra. Con esta técnica se eliminan los posibles retardos asociados a las conmutaciones entre hebras, como se han descrito anteriormente. No obstante, el número de instrucciones que se emite en cada ciclo sigue estando limitado por las dependencias que existen dentro de una hebra.
- **Superescalar multihebra con bloqueo** → De nuevo, solo instrucciones de una hebra pueden emitirse en un ciclo, y se utiliza procesamiento multihebra con bloqueo.

- **Procesadores de palabra de instrucciones muy larga (VLIW – *Very Long Instruction Word*)** → Una arquitectura VLIW, tal como la de la IA-64, ubica varias instrucciones en una única palabra. Usualmente, el compilador construye los códigos VLIW ubicando operaciones que pueden ejecutarse en paralelo dentro de la misma palabra. En una máquina VLIW sencilla (Figura 6.16 (g)) si no es posible rellenar completamente la palabra con instrucciones que pueden emitirse en paralelo, se utilizan códigos de no operar (*no-ops*).
- **VLIW multihebra con entrelazado** → Esta aproximación proporcionaría eficiencias similares a la del procesamiento multihebra con entrelazado en una arquitectura superescalar.
- **VLIW multihebra con bloqueo** → Esta aproximación proporcionaría eficiencias similares a la del procesamiento multihebra con bloqueo en una arquitectura superescalar.

Las dos últimas aproximaciones que se muestran en la Figura 6.16 permiten la ejecución simultánea y paralela de varias hebras:

- **Procesadores multihebra simultánea (SMT)**→ La Figura 6.16 (j) muestra un sistema que es capaz de emitir ocho instrucciones a la vez. Si una hebra tiene un mayor grado de paralelismo entre instrucciones podría, en algunos ciclos, llenar todos los huecos de emisión horizontales. En otros ciclos, se podrían emitir instrucciones de dos o más hebras. Si hay suficientes hebras activas, usualmente sería posible emitir el máximo número de instrucciones en cada ciclo, proporcionando un alto nivel de eficiencia.
- **Multiprocesadores monochip** → La Figura 6.16 (k) correspondería a un chip con cuatro procesadores, cada uno de los cuales es un procesador superescalar capaz de emitir dos instrucciones. A cada uno de los procesadores se le asigna una hebra, de la que puede emitir hasta dos instrucciones por ciclo.

Comparando las figuras 6.16 (j) y 6.16 (k), vemos que un multiprocesador monochip con la misma capacidad de emisión que un SMT no es capaz de conseguir el mismo grado de paralelismo entre instrucciones. Esto es debido a que el procesador monochip no puede ocultar las latencias emitiendo instrucciones de otras hebras. No obstante, el multiprocesador monochip mejoraría las prestaciones de un superescalar con la misma capacidad de emisión de instrucciones, debido a que las pérdidas horizontales serían mayores en el procesador superescalar. Además, es posible utilizar procesamiento multihebra dentro de cada uno de los procesadores del multiprocesador monochip, y esto es lo que se hace en algunas de las máquinas actuales.

EJEMPLOS DE SISTEMAS

- **Pentium 4** → Los últimos modelos del Pentium 4 utilizan una técnica de procesamiento multihebra que, en la literatura de Intel, se denomina *hyperthreading*. Básicamente la aproximación del Pentium 4 corresponde a un SMT que permite procesar dos hebras; de esta forma, un único procesador multihebra equivaldría a dos procesadores lógicos.
- **IBM Power5** → El chip Power5 de IBM, utilizado en los productos PowerPC de gama alta, combina multiprocesamiento monochip con SMT. El chip tiene dos procesadores, cada uno de los cuales es un procesador multihebra capaz de procesar dos hebras concurrentemente utilizando SMT. Hay que resaltar que, los

diseñadores simularon varias alternativas y encontraron que dos procesadores SMT de dos vías en un chip proporcionaban mejores prestaciones que un solo procesador SMT de cuatro vías. Las simulaciones pusieron de manifiesto que el procesamiento multihebra de más de dos hebras podría ocasionar una reducción de prestaciones debido a la interacción por los accesos a caché, dado que los datos que necesita una hebra podrían desplazar de caché a los datos que necesita otra hebra.

La Figura 6.17 muestra el diagrama de flujo de instrucciones del Power5. Sólo deben repetirse algunos de los elementos del procesador para disponer de elementos dedicados a cada una de las hebras. Se utilizan dos contadores de programa. El procesador alterna la captación de instrucciones (hasta ocho cada vez) entre las dos hebras. Todas las instrucciones se almacenan en una caché de instrucciones común y comparten un recurso de traducción de instrucciones que lleva a cabo una decodificación de instrucciones parcial. Cuando aparece un salto condicional, el recurso de predicción de salto predice la dirección del mismo y, si es posible, calcula la dirección de destino del salto. Para predecir el destino de un retorno de subrutina, el procesador utiliza una pila (pila de retorno) para cada hebra.

Después de la decodificación de instrucciones parcial, las instrucciones pasan a dos buffers de instrucciones separados; entonces, según la prioridad de la hebra, se selecciona un grupo de instrucciones y se decodifican en paralelo. A continuación, las instrucciones pasan a través de un recurso de renombramiento de registros en el mismo orden en el que se encuentran en el programa y se asignan registros lógicos a los registros físicos. El Power5 tiene 120 registros físicos de propósito general y 120 registros de coma flotante. Luego las instrucciones pasan a las colas de emisión; desde estas colas, las instrucciones se emiten utilizando multihebra simétrica. Es decir, el procesador tiene una arquitectura superescalar que puede emitir instrucciones de las dos hebras en paralelo. Al final del cauce, existen recursos separados para cada una de las hebras, necesarios para la finalización de las instrucciones.

6.5 CLUSTERS

Los *clusters* constituyen la alternativa a los *multiprocesadores simétricos* (SMP) para disponer de prestaciones y disponibilidad elevadas, y son particularmente atractivos en aplicaciones propias de un servidor.

Un ***cluster*** se define como un grupo de *computadores completos* interconectados que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola máquina. El término *computador completo* hace referencia a un sistema que puede funcionar por si solo, independientemente del *cluster*. Usualmente, en la literatura, cada computador del cluster se denomina **nodo**.

Los cuatro beneficios que pueden conseguirse con un *cluster*, que además van a ser los objetivos o requisitos de diseño son los siguientes:

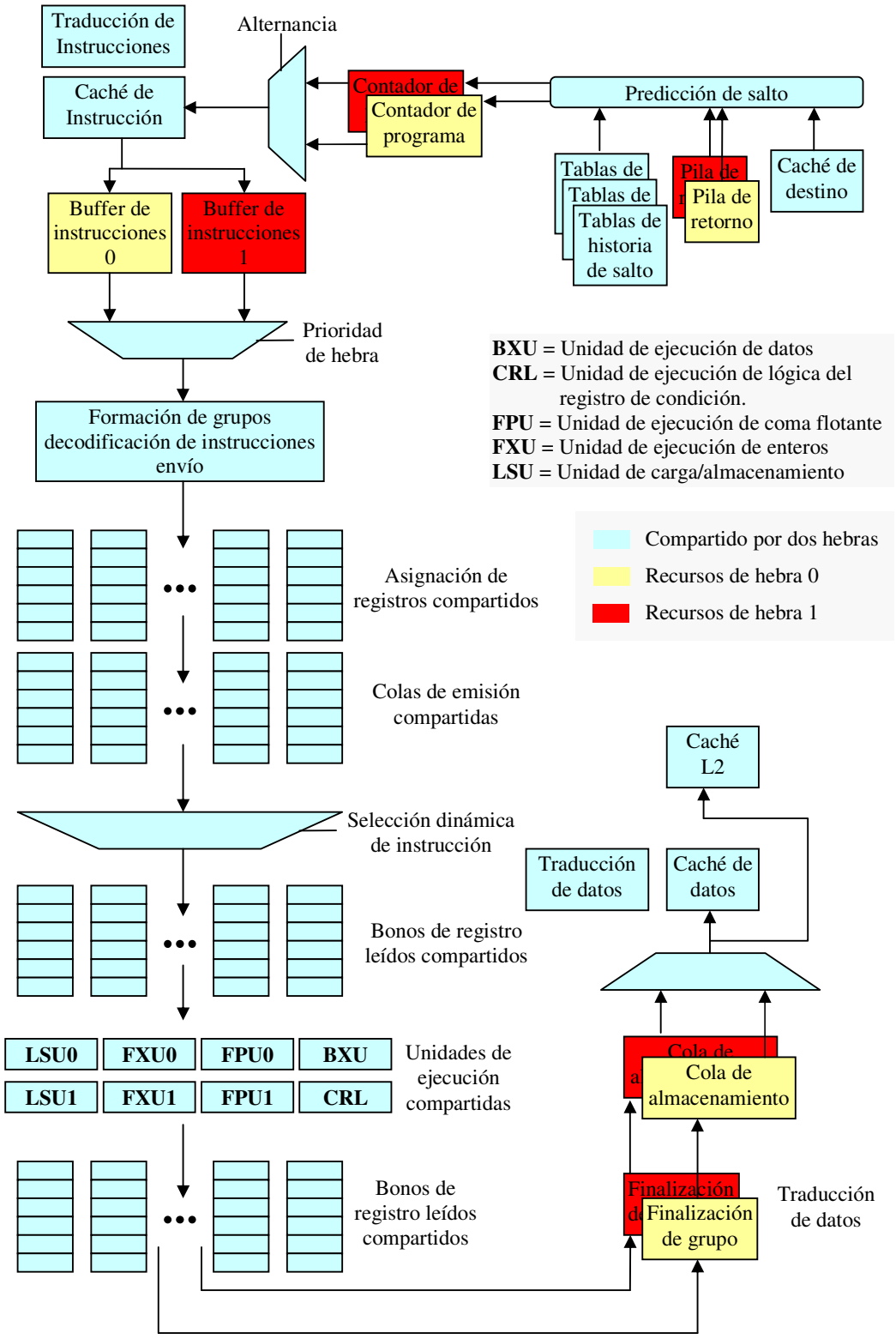


Figura 6.17 Flujo de instrucciones en el Power5.

- **Escalabilidad absoluta** → Es posible configurar *clusters* grandes que incluso superan las prestaciones de los computadores independientes más potentes. Un *cluster* puede tener decenas de máquinas, cada una de las cuales puede ser un multiprocesador.
- **Escalabilidad incremental** → Un *cluster* se configura de forma que sea posible añadir nuevos sistemas al *cluster* en aplicaciones sucesivas. Así, se puede comenzar con un sistema modesto y ampliarlo a medida que se vaya necesitando, sin tener que sustituir el sistema de que dispone por uno nuevo que proporcione mayores prestaciones.
- **Alta disponibilidad** → Puesto que cada nodo del *cluster* es un computador autónomo, el fallo de uno de los nodos no significa la pérdida del servicio. En muchos casos es el software el que proporciona automáticamente la tolerancia a fallos.
- **Mejor relación precio-prestaciones** → Al utilizar elementos estandarizados, es posible configurar un *cluster* con mayor o igual potencia de cómputo que un computador independiente mayor, y a mucho menos coste.

CONFIGURACIONES DE CLUSTERS

En la literatura se pueden encontrar clasificaciones diversas de los *clusters*. Quizás la clasificación más sencilla es la que considera si compartir o no el acceso al mismo disco. La Figura 6.18 (a) muestra un *cluster* de dos nodos en el que la interconexión se realiza mediante un enlace de alta velocidad que puede utilizarse para intercambiar mensajes que coordinan la actividad del *cluster*. El enlace puede ser una LAN que se comparte con otros computadores no incluidos en el *cluster*, o puede tratarse de un medio de interconexión específico. En este último caso, uno o varios computadores del *cluster* tendrán un enlace a una LAN o a una WAN de forma que sea posible la conexión entre el *cluster*, actuando como servidor, y los clientes remotos. En la Figura 6.18, cada computador se representa como multiprocesador; esto no es imprescindible.

En la Figura 6.18 (b) se muestra otra alternativa, ahora con disco compartido. Este subsistema de disco común es un RAID (*Redundant Array of Independent Disks*). El uso del RAID o algún tipo de tecnología redundante es común en los *clusters* para que la elevada disponibilidad que se consigue con la presencia de varios computadores no se vea comprometida por un disco compartido que pueda convertirse en el único punto de fallo.

El abanico de posibilidades de configuración de *clusters* es más amplio si se tiene en cuenta las alternativas funcionales (Tabla 6.2).

- **Espera pasiva (*passive standby*)** → Este método es bastante antiguo y común, y simplemente consiste en mantener toda la carga de trabajo en un computador mientras que otro permanece inactivo hasta que se produzca un fallo y tome el relevo del primero. Para coordinar las máquinas, el computador activo, o primario, envía un "mensaje de actividad (*heartbeat message*)" al computador en espera. En el caso de que estos mensajes dejen de llegar, el computador en espera asume que el servidor ha fallado y se pone en marcha. Esta alternativa aumenta la disponibilidad pero no las prestaciones; es más, si los dos computadores solo se intercambian el mensaje de actividad, y si no tienen discos comunes, entonces el computador en espera constituye un computador de

reserva para ejecutar procesos pero no tiene acceso a la base de datos gestionada por el primer computador.

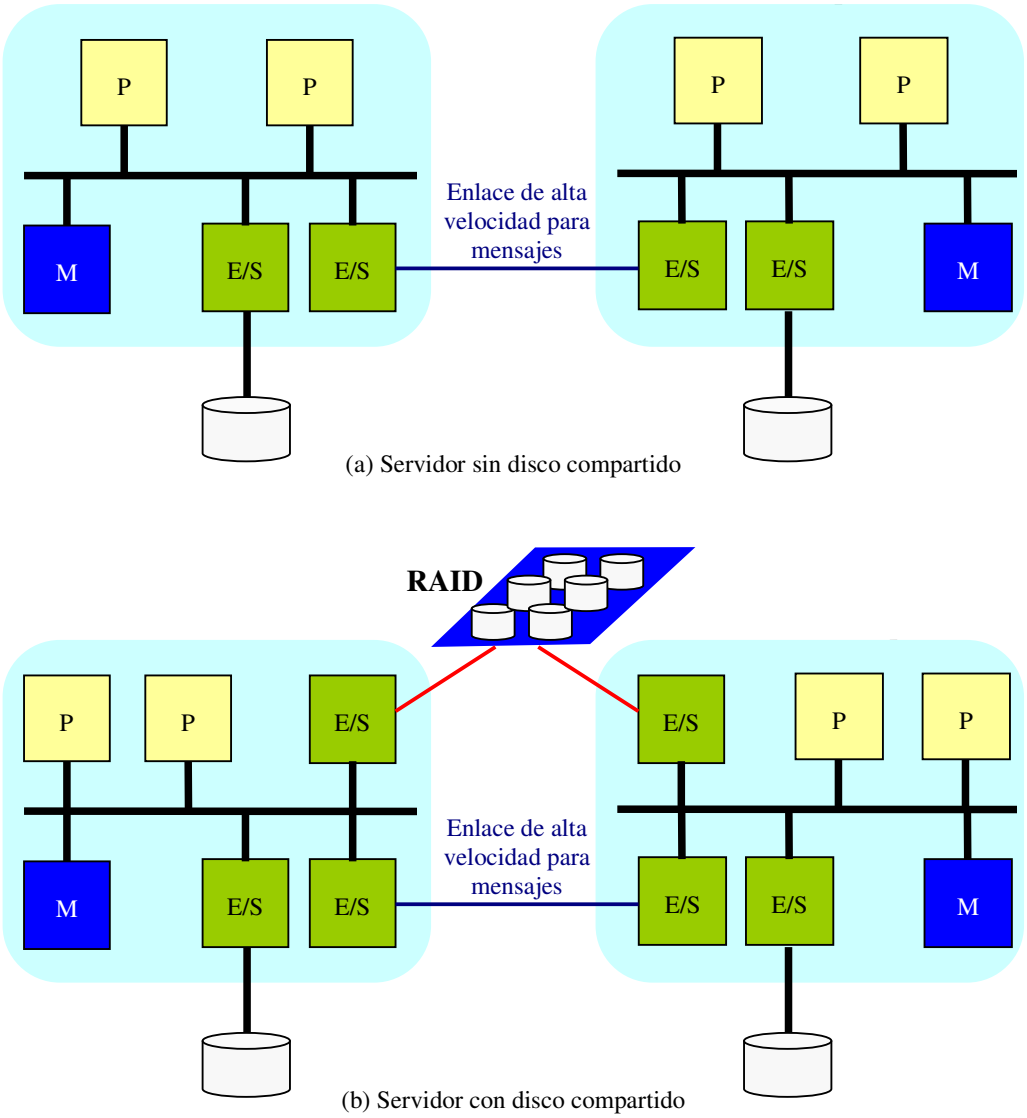


Figura 6.18 Configuraciones de *cluster*.

- **Secundario activo (*active secondary*)** → La espera activa no suele aplicarse en un *cluster*. El término *cluster* hace referencia a varios computadores interconectados que se encuentran activos realizando algún tipo de procesamiento, a la vez que proporcionan una imagen cara al exterior de sistema único. El término secundario activo se utiliza a menudo para referirse a esta configuración. Se pueden distinguir tres métodos para componer el *cluster*:

  - **Servidores separados (separate server)** → Cada computador es un servidor independiente, con su propio disco y no existen discos compartidos por los sistemas (Figura 6.18 (a)). Esta organización proporciona tanto una disponibilidad como prestaciones elevadas. Se

precisa algún tipo de software de gestión o planificación para asignar a los servidores las peticiones que se van recibiendo de los clientes, de forma que se equilibre la carga de los mismos y se consiga una utilización elevada. Es deseable tener una cierta capacidad de fallo (si un computador falla mientras está ejecutando una aplicación, otro computador del *cluster* puede acceder a la aplicación y completarla); para que esto sea posible, los datos se deben copiar constantemente en los distintos sistemas, de manera que cada procesador pueda acceder a los datos actuales de los demás. El coste que supone este intercambio de datos ocasiona una penalización en las prestaciones; este es el precio por conseguir una disponibilidad elevada.

- **Servidores conectados a discos** → Para reducir el coste que suponen las comunicaciones, la mayoría de los *cluster* están constituidos por servidores conectados a discos comunes (Figura 6.18 (b)). Una variación a esta alternativa se designa como configuración **sin compartir nada** (***shared nothing***). En esta aproximación, los discos comunes se dividen en volúmenes diferentes y cada volumen pasa a pertenecer a un solo procesador; si el computador falla, el *cluster* se debe reconfigurar de forma que los volúmenes que pertenecían al computador que ha fallado pasen al resto de computadores.
- **Servidores compartiendo discos** → Es posible que los computadores compartan el disco al mismo tiempo, para que todos tengan acceso a todos los volúmenes de todos los discos. Esta aproximación necesita utilizar algún tipo de procedimiento para el acceso exclusivo (para asegurar que en un momento dado, solo un computador puede acceder a los datos)

Configuración del Cluster	**Descripción**	**Beneficios**	**Limitaciones**
Espera pasiva	Un servidor secundario sustituye al servidor primario en caso de que falle	Fácil de implementar	Alto coste, debido a que el servidor secundario no está disponible para otras tareas de procesamiento
Secundario activo	El servidor secundario también se utiliza para tareas de procesamiento	Coste reducido porque los servidores secundariso también se emplean para el procesamiento	Aumenta la complejidad
Seridores separados	Cada servidor tiene su propio disco. Los datos se copian desde el servidor primario al secundario	Alta disponibilidad	Penalización elevada en la red y el servidor debido a las operaciones de copia
Servidores conectados a discos	Aunque hay conexión a unos discos comunes, cada servidor también tiene sus propios discos. Si falta un servidor, otro servidor accede a sus discos	Penalización reducida en la red y en el servidor debido a la eliminación de las operaciones de copia	Usulamente se necesitan discos espejo o tecnología RAID para afrontar el riesgo de fallo de disco
Servidores compartiendo discos	Los servidores comparten simultáneamente el acceso a los discos	Penalización baja en la red y en el servidor. Riesgo reducido de fallo del sistema debido a un fallo de disco	Se necesita software de control de acceso exclusivo. Usualmente se utiliza con discos espejo o tecnología RAID

Tabla 6.2 Métodos de configuración de *clusters*; beneficios y limitaciones.

CONSIDERACIONES EN EL DISEÑO DEL SISTEMA OPERATIVO

Un completo aprovechamiento de la configuración hardware de un cluster exige una cierta ampliación de un sistema operativo propio de un único computador:

- **Gestión de fallos** → La forma en que se actúa frente a un fallo en un *cluster* depende del tipo de configuración del *cluster* (Tabla 6.2). En general se pueden utilizar dos alternativas para enfrentarse a los fallos:

  - **Clusters de alta disponibilidad** → Es el que ofrece una probabilidad elevada de que todos sus recursos estén en servicio. Si se produce un fallo, tal como la caída del sistema o la pérdida de un volumen de disco, se pierden las tareas en curso. Si una de esas tareas se reinicia, puede ser un computador distinto el que le dé servicio. No obstante el sistema operativo del *cluster*, no garantiza el estado de las transacciones ejecutadas parcialmente; esto debería gestionarse en el nivel de aplicación.
  - **Clusters tolerantes a fallos** → Este garantiza que todos los recursos estén disponibles. Esto se consigue utilizando discos compartidos redundantes y mecanismos para salvar las transacciones no terminadas y concluir las transacciones terminadas.

  La función de conmutar aplicaciones y datos en el *cluster*, desde un sistema defectuoso a otro alternativo, se denomina *transferencia por fallo* (*failover*). Una función adicional es la restauración de las aplicaciones y los datos por el sistema original, una vez superado el fallo; se denomina *recuperación después de un fallo* (*failback*). La recuperación puede hacerse automáticamente, pero esto es deseable sólo si el fallo ha sido completamente reparado y es poco probable que vuelva a producirse. En caso contrario, la recuperación automática puede ocasionar transferencias continuas de programas y datos, en un sentido y en otro, debido a la reaparición de un fallo; y se producirían problemas en cuanto a las prestaciones y a la recuperación.

- **Equilibrado de carga** → Un *cluster* necesita de una capacidad efectiva para equilibrar la carga entre los computadores disponibles. Esta capacidad es necesaria para satisfacer el requisito de la escalabilidad incremental. Cuando un computador se añade al *cluster*, las aplicaciones encargadas de la asignación de tareas deberían incluir automáticamente a dicho computador junto con los restantes, para distribuir la carga de forma equilibrada. Los mecanismos de un nivel de software intermedio entre el sistema operativo y las aplicaciones (*middleware*) necesitan reconocer los servicios que pueden aparecer en los distintos miembros del *cluster* y pueden migrar desde un miembro a otro.

- **Computación paralela** → En algunos casos, la utilización eficiente de un *cluster* requiere ejecutar en paralelo el software correspondiente a una aplicación. A continuación muestran tres posibles soluciones al problema:

  - **Paralelización mediante el compilador** → Un compilador con capacidad de generación de código paralelo determina en el momento de

la compilación las partes de la aplicación que pueden ejecutarse en paralelo. Cada una de estas partes se asignan a computadores distintos del *cluster*. Las prestaciones dependen de la naturaleza del problema y de cómo esté diseñado el compilador.

- **Paralelización realizada por el programador** → El programador escribe la aplicación para que se ejecute en el *cluster*, y utiliza paso de mensajes para mover los datos entre los nodos del *cluster* según sea necesario. Aunque aumenta la carga de trabajo del programador, puede ser la mejor aproximación para sacar partido a un *cluster* en algunas aplicaciones.
- **Computación paramétrica** → Se puede utilizar esta aproximación si la aplicación consiste básicamente en un algoritmo o programa que debe ejecutarse una gran cantidad de veces, pero cada vez con un conjunto diferente de condiciones de inicio o parámetros; por ejemplo, un modelo de simulación, que debe ejecutarse para un gran número de escenarios de forma que se obtengan resultados estadísticamente significativos. Para que esta aproximación sea efectiva se necesitan herramientas de procesamiento paramétrico que organicen, gestionen y envíen a ejecutar los trabajos de forma ordenada.

ARQUITECTURA DE LOS CLUSTERS

En la Figura 6.19 se muestra una arquitectura típica de *cluster*. Los computadores se conectan a través de una red de área local (LAN) de alta velocidad o mediante un conmutador. Cada computador puede trabajar de forma independiente; además, en cada computador se instala una capa software intermedia (*middleware*) que permite el funcionamiento de todos los computadores como un único computador (*cluster*). El *middleware* del *cluster* proporciona al usuario una imagen unificada conocida como **imagen de sistema único** (***single-system image***); el *middleware* es también responsable de proporcionar alta disponibilidad, distribuyendo la carga y respondiendo a los fallos de los componentes. Los servicios y funciones deseables en la capa de *middleware* son:

- **Punto de entrada único** → Un usuario entra en el *cluster* igual que si entrara en un único computador.
- **Jerarquía de ficheros única** → El usuario ve una sola jerarquía de directorios de ficheros bajo el mismo directorio raíz.
- **Punto de control único** → Por defecto, existe un único computador que se utiliza para la gestión y control del *cluster*.
- **Red virtual única** → Un nodo puede acceder a cualquier otro nodo del *cluster*, incluso aunque la configuración presente en el *cluster* conste de varias redes interconectadas. Existe una única operación de red virtual.
- **Espacio de memoria único** → La memoria compartida distribuida permite que los programas compartan variables.
- **Sistema de gestión de trabajo único** → El planificador de trabajos de un *cluster* permite que un usuario pueda enviar un trabajo sin especificar el computador donde se va a ejecutar el trabajo.
- **Interfaz de usuario única** → Existe una interfaz gráfica común para todos los usuarios, independientemente del computador desde el que se entre al *cluster*.

- **Espacio de E/S único** → Cualquier nodo puede acceder remotamente a cualquier periférico de comunicación o de disco sin que conozca su ubicación física.
- **Espacio de procesos único** → Se utiliza un esquema de identificación de procesos uniforme. Un proceso de cualquier nodo puede crear o comunicarse con cualquier otro proceso en un nodo remoto.
- **Puntos de chequeo** → El estado del proceso y los resultados intermedios se almacenan periódicamente para permitir la recuperación después de un fallo.
- **Migración de procesos** → Con el fin de permitir la distribución equilibrada de la carga.

Los cuatro últimos puntos de la lista anterior mejoran la disponibilidad del *cluster*; los restantes contribuyen a la imagen de sistema único.

Además, que también se representa en la Figura 6.19, un cluster también deberá incluir herramientas software para permitir una ejecución eficiente de los programas que pueden ejecutarse en paralelo.

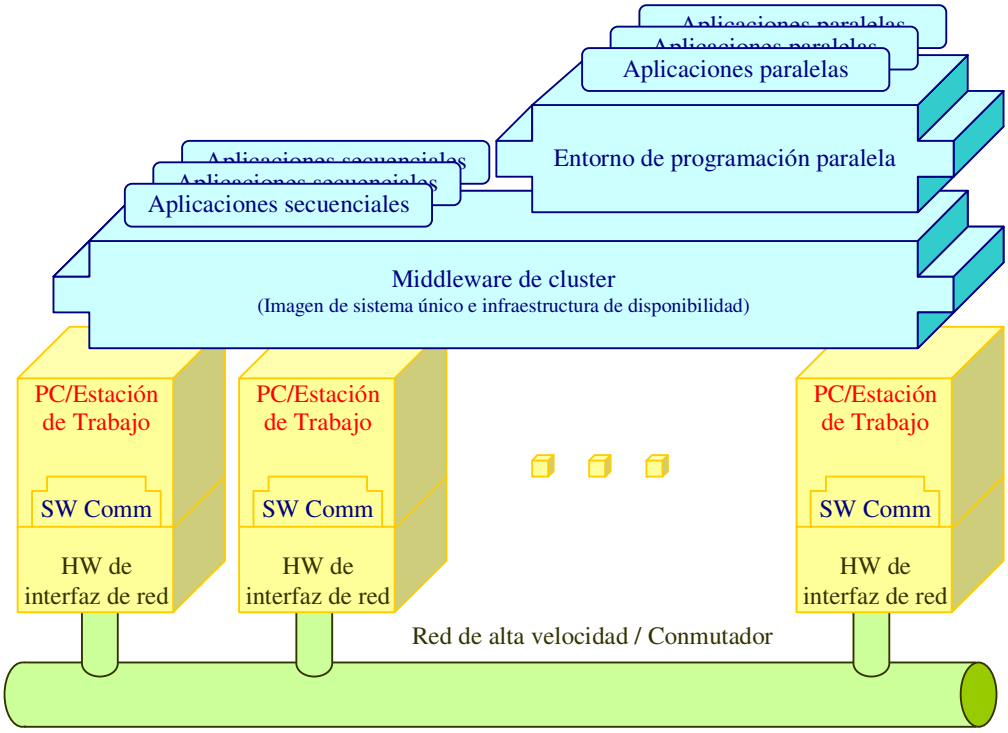


Figura 6.19 Arquitectura de computador de un *cluster*.

CLUSTERS FRENTE A SISTEMA SMP

Tanto los *clusters* como los multiprocesadores simétricos (SMP) constituyen configuraciones con varios procesadores que pueden ejecutar aplicaciones con una alta demanda de recursos. Ambas soluciones están disponibles comercialmente, aunque los SMP lo están desde hace más tiempo.

La principal ventaja de un SMP es que resulta más fácil de gestionar y configurar que un *cluster*. El SMP está mucho más cerca del modelo de computador de un solo procesador para el que están disponibles casi todas las aplicaciones; el principal cambio que se necesita para pasar de un computador monoprocesador a un SMP se refiere al funcionamiento del planificador. Otra ventaja de un SMP es que necesita menos espacio físico y consume menos energía que un *cluster* comparable. Una última e importante ventaja para los SMP es que son plataformas estables y bien establecidas.

Los *clusters* son superiores a los SMP en términos de escalabilidad absoluta e incremental, y además también son superiores en términos de disponibilidad, puesto que todos los componentes del sistema pueden hacerse altamente redundantes.

6.6 OTRAS FORMAS DE SUPERCOMPUTACIÓN. COMPUTACIÓN GRID

Se llama **Computación Grid** (***Grid Computing***) a un sistema de computación distribuido que permite compartir recursos no centrados geográficamente para resolver problemas de gran escala.

Cuando Charles Babbage diseñó la primera computadora programable a mediados del sigo XIX, lo hizo con el propósito de resolver eficientemente un problema contemporáneo: la tabulación de fórmulas matemáticas que, hasta entonces, eran calculadas manualmente por operadores humanos (llamados «computadores»). Babbage fue un pionero de la informática, y sus ideas permanecieron relativamente en la oscuridad hasta que, más de 100 años después, se construyeron los primeros ordenadores electrónicos. Nuevamente, la principal motivación fue un problema especialmente importante en esos tiempos: descifrar los códigos de la Alemania nazi durante la Segunda Guerra Mundial.

Desde entonces, la informática ha evolucionado mucho, y también lo han hecho los problemas que deben resolver los ordenadores. En lugar de tabulaciones matemáticas o mensajes cifrados, ahora los ordenadores se enfrentan a nuevas tareas, como el análisis de las partículas más fundamentales del universo y complicadas simulaciones de modelos medioambientales. Para abordar estos problemas, se construyen enormes supercomputadores, miles de veces más potentes que nuestro ordenador de sobremesa, capaces de realizar billones de operaciones por segundo. Sin embargo, nuestro afán por el conocimiento, por el planteamiento de nuevos y emocionantes problemas que resolver, evoluciona más rápidamente que la informática, y a veces nos encontramos con que estos gigantes computacionales resultan insuficientes y hay que saltar al siguiente nivel: la Computación Grid.

Para explicar lo que es la Computación Grid conviene utilizar un ejemplo de un problema de proporciones gigantescas. Un buen ejemplo, que a estas alturas se ha convertido en el «embajador de la Grid» para los no-iniciados, es el Large Hadron Collider (LHC), un acelerador de partículas con una circunferencia de 27 km cuya construcción se ha realizado en el CERN (el Centro Europeo de Investigaciones Nucleares), cerca de Ginebra. Los experimentos que se realizan en el acelerador producen alrededor de 15 petabytes de información cada año (un petabyte equivale a 1.048.576 gigabytes; un disco duro típico de sobremesa suele rondar los 150-300 gigabytes). Esta información tiene que ser almacenada y procesada, y eso resulta

inabordable en un único centro de cálculo, ya que sería necesario equipar al CERN con una potencia de cómputo varios órdenes de magnitud mayor de la actualmente disponible, una opción impracticable tanto a nivel técnico como económico.

La Computación Grid es un modelo de computación relativamente nuevo que plantea repartir el cómputo de estos problemas entre recursos computacionales (como supercomputadores, clusters de ordenadores, etc.) distribuidos en diversas organizaciones. Conceptualmente, es una solución muy sencilla, un «divide y vencerás» de toda la vida: si no podemos resolver el problema en un único lugar (Ginebra), pues lo repartimos entre centros de cálculo de toda Europa, cuya unión sí da cabida a las necesidades computacionales del LHC. Esta unión de recursos de distintas organización es lo que habitualmente se denomina una «grid» computacional.

Sin embargo, en la práctica, realizar este «reparto» es técnicamente muy complicado. Por ejemplo, supongamos que queremos distribuir un enorme cómputo entre centros de cálculo situados en distintos países. ¿Cómo «dividimos» el problema? Y una vez dividido, ¿cómo realizamos el reparto de las distintas subtareas? ¿Cómo conseguimos que los distintos centros de cálculo (cada uno con distintos tipos de máquinas, redes, sistemas operativos, etc.) se comuniquen entre ellos? Desde el punto de vista de alguien que cede parte de sus recursos a una grid, ¿cómo garantizamos que nadie abusa de esos recursos que se están cediendo? ¿Cómo mover las enormes cantidades de datos de un centro a otro? La Computación Grid se encarga de responder a estas y muchas otras preguntas.

Por supuesto, dicho todo esto, puede parecer que la Computación Grid no es más que pura fantasía, o algo que difícilmente puede afectar al público general (al fin y al cabo, no es habitual tener un acelerador de partículas en el salón de casa...). Sin embargo, hace tiempo que la Computación Grid dejó de ser una tecnología experimental, y actualmente ya se utiliza para mejorar la calidad investigadora de muchos proyectos en todo el mundo. El LHC, por ejemplo, se vale de una grid europea, fruto del proyecto EGEE, que une los recursos computacionales de más de 100 centros de cálculo alrededor de todo el mundo (la mayoría situados en Europa).

Eso sí, la Computación Grid no se limita a resolver enormes problemas, como los planteados por el LHC. También puede proporcionar potencia computacional sobre demanda a proyectos más modestos. De hecho, el término «Grid» se originó por analogía a la red eléctrica («Electrical Grid»). Cuando nosotros necesitamos electricidad, evidentemente no necesitamos contar con nuestro propio generador eléctrico.

Sencillamente nos conectamos a la red eléctrica, que nos proporcionará la potencia que nos hace falta, y que se habrá originado en una central eléctrica que atiende las necesidades de muchos usuarios. La Computación Grid, en esencia, permite que un usuario tenga acceso a toda la potencia computacional que necesita (la «electricidad»), pero sin tener que disponer de un supercomputador propio (la «central eléctrica»). Así pues, hay muchas Grids alrededor del mundo, tales como EGEE (en Europa) y TeraGrid (en EEUU) que proporcionan una valiosa infraestructura a distintos proyectos relacionados con bioinformática, astrofísica, química computacional, finanzas y muchas otras áreas. Las grids como EGEE y TeraGrid afectan al público general indirectamente al mejorar los recursos de los que disponen científicos que trabajan en

problemas que, de una manera u otra, acaban afectándonos. Sin embargo, también hay grids centradas en solucionar problemas que afectan más directamente a la sociedad, como caBIG (Cancer Biomedical Informatics Grid), una Grid para aunar recursos en la lucha contra el cáncer, y NEES (Network for Earthquake Engineering Simulation), una red de investigación que utiliza una infraestructura grid para la simulación de terremotos.

La Computación Grid, a pesar de ser una tecnología con bastante madurez, sigue estando en constante evolución. Actualmente existen muchas grids computacionales (como EGEE y TeraGrid), construidas para dar salida principalmente a problemas científicos, pero todavía no existe «La Grid». De la misma manera que Internet nació en el ámbito científico para luego llegar al público general, lo mismo puede esperarse, a largo plazo, de la Computación Grid. Cuando la tecnología madure lo suficiente, será posible que cualquier usuario, desde su ordenador personal de casa, pueda enviar complejos trabajos computacionales a «La Grid», como si tuviésemos un supercomputador en el salón de casa.

Autor: Borja Sotomayor (Departamento de Ciencias de la Computación de la Universidad de Chicago)

Existen varios conceptos acerca de qué es un *grid*. Uno de ellos, elaborado por el *Grid Computing Information Centre*, una de las asociaciones dedicada exclusivamente al desarrollo de esta tecnología, llama ***grid*** a "*un tipo de sistema paralelo y distribuido que permite compartir, seleccionar y reunir recursos autónomos geográficamente distribuidos en forma dinámica y en tiempo de ejecución, dependiendo de su disponibilidad, capacidad, desempeño, coste y calidad de servicio requerida por sus usuarios*". Según esta definición, se busca aprovechar la sinergia que surge de la cooperación entre recursos computacionales y proveerlos como servicios.

Otra definición más estructurada, expuesta por Foster, Kesselman y Tuecke, precursores de la *computación grid*, plantea la existencia de *organizaciones virtuales* (OV) como puntos de partida de este enfoque. Una *organización virtual* es "*un conjunto de individuos y/o instituciones definida por reglas que controlan el modo en que comparten sus recursos*". Básicamente, son organizaciones unidas para lograr objetivos comunes. Ejemplos de OVs podrían ser proveedores de servicios de aplicaciones o almacenamiento, equipos de trabajo empresarial realizando análisis y planeamiento estratégico, miembros de una planta de energía evaluando trabajo de campo, universidades involucradas en un proyecto de investigación conjunto, etc. Las OVs varían enormemente en cuanto a sus objetivos, alcance, tamaño, duración, estructura, comunidad y sociología. Sin embargo, existen varios requerimientos y problemas subyacentes tales como la necesidad de relaciones flexibles para compartir recursos, niveles de control complejos y precisos, variedad de recursos compartidos (programas, archivos, datos, sensores y redes, entre otros), modos de funcionamiento (individual, multiusuario), calidad de servicio, etc. Las tecnologías actuales o bien no proveen espacio para la variedad de recursos involucrados o no aportan la flexibilidad y control de las relaciones cooperativas necesarias para establecer las Ovs.

Como solución, se propone el grid como un modelo de trabajo para "compartir recursos en forma coordinada y resolver problemas en organizaciones virtuales multi-institucionales de forma dinámica". De esta manera, varias instituciones pueden formar

distintas OVs e incluso formar parte de más de una al mismo tiempo, realizando diferentes roles e integrando distintos recursos como se muestra en la Figura 6.20.

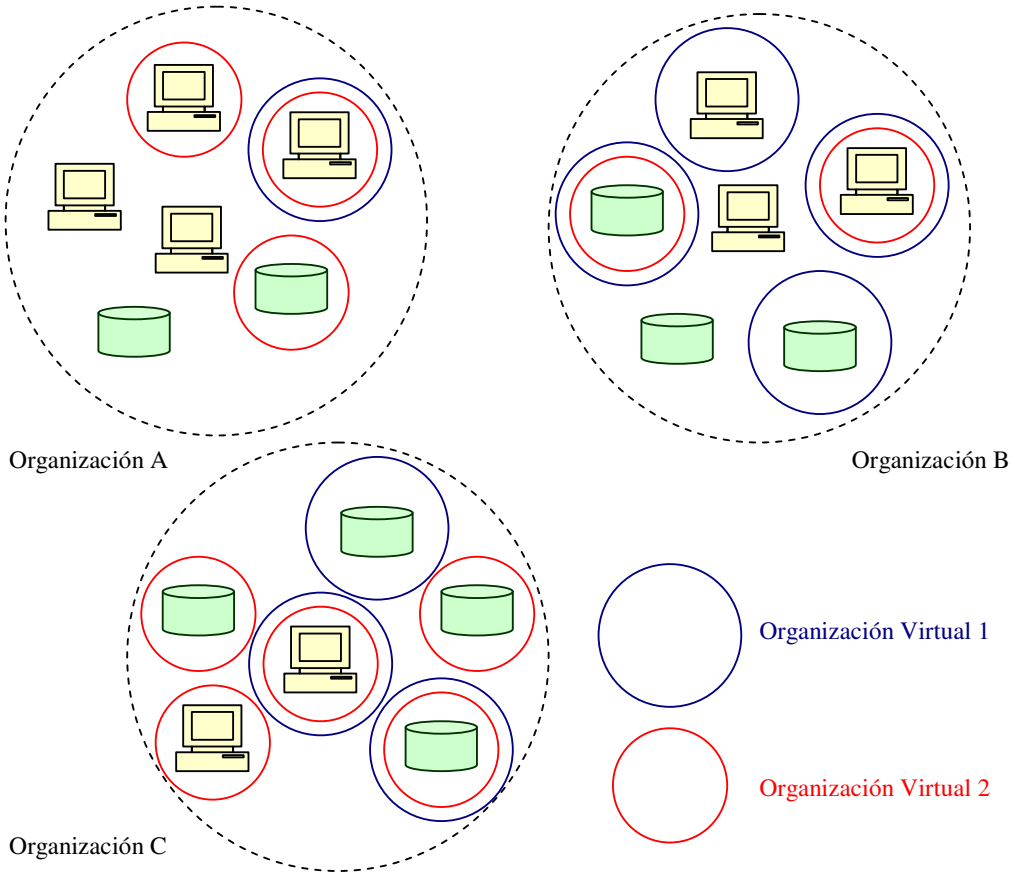


Figura 6.20 Una organización real puede participar en una o más OVs compartiendo algunos o todos sus recursos. Se muestran tres organizaciones reales (los círculos de puntos) y dos OVs (conformadas por los recursos compartidos en cada caso). Las políticas que controlan el acceso a los recursos varían de acuerdo a las organizaciones reales, los recursos y las OVs involucradas. Nótese que existen recursos que componen una o más OVs y otros que no se comparten.

Para lograr este nuevo cometido se han desarrollado varios protocolos, servicios y herramientas que intentan sustentar organizaciones virtuales escalables. Debido a su enfoque basado en compartir recursos de manera dinámica y multi-organizacional, las tecnologías grid se complementan en vez de competir con las tecnologías de computación distribuida existentes.

DEFINICIONES

Sharing (uso compartido, puesta en común, intercambio) → El sharing de los recursos es condicional: cada dueño de los recursos hace disponible el mismo sujeto a condiciones de cuándo, dónde, y qué es lo que puede realizarse. Los consumidores de recursos pueden colocar condiciones en propiedades de los recursos para los cuales ellos están preparados para trabajar.

Las relaciones de sharing pueden variar dinámicamente sobre el tiempo en términos de recursos involucrados, la naturaleza de los accesos permitidos y los participantes que tienen accesos permitidos. Estas no son a menudo simplemente de tipo cliente-servidor, por *peer to peer* (de igual a igual, de punto a punto, de par a par, P2P – red informática entre iguales): los proveedores pueden ser clientes y las relaciones de sharing pueden existir sobre un subconjunto de participantes. Además pueden combinarse de forma coordinada sobre muchos recursos, cada uno perteneciente a diferentes organizaciones. La habilidad de delegar en vías controladas se torna importante en algunas situaciones como hacer mecanismos para coordinar operaciones a través de múltiples recursos.

Un mismo recurso puede usarse de diversas formas dependiendo de las restricciones situadas en el lugar desde donde se comparten y sus objetivos. Por ejemplo una computadora puede usarse solamente para correr piezas de software específicas en un arreglo de sharing mientras que éste puede proveer ciclos genéricos de operación en otra.

En el contexto de una red la interoperabilidad se refiere a protocolos comunes. Nuestra Arquitectura Grid es primeramente una arquitectura de protocolos que definen los mecanismos básicos para cada usuario de las OV y la negociación de recursos, establecimiento, manejo y la utilización de las relaciones de sharing. La arquitectura basada en estándares facilita la extensibilidad, interoperabilidad, portabilidad y el sharing de código; los protocolos estándares hacen más fácil definir servicios estándares que provean capacidades. Se pueden construir interfases de aplicaciones para programación y kits de desarrollo de software que provean abstracciones de programación requeridas para crear una Grid usable.

A través de la interoperabilidad se asegura que las relaciones de sharing puedan iniciarse en partes arbitrarias, acomodándose dinámicamente a nuevos participantes sobre diferentes plataformas, lenguajes y entornos de desarrollo.

La definición de un protocolo especifica cómo los elementos distribuidos de sistemas interactúan en orden de devolver un comportamiento específico y la estructura del intercambio de información durante su interacción.

Las OV tienden a fluir y se notan ciertos cambios: los mecanismos usados localizan recursos y se establecen y reconocen identidades, por ello la determinación de autorizaciones e iniciación del sharing debe ser flexible y liviano de forma que el arreglo de sharing de recursos pueda establecerse y cambiarse fácilmente.

Servicios → Un servicio se define por el protocolo con el que habla y el comportamiento que implementa. La definición de servicio estándar (de acceso a computación, datos, descubrimiento de recursos, replicación de datos y más) nos permite mejorar los servicios ofrecidos a participantes de las OV y también abstraer de detalles específicos de los recursos que de otra forma deberían ser posteriores al desarrollo de aplicaciones para OV.

ARQUITECTURA GRID

La arquitectura grid estándar propuesta en la Figura 6.21 identifica componentes fundamentales del sistema y además las principales funciones y propósitos que deberían cumplir las aplicaciones, toolkits APIs, para componer un sistema grid.

Dicha arquitectura es una arquitectura abierta, donde los componentes del sistema se organizan en capas dentro de las cuales comparten características similares. La Figura 6.21 muestra la arquitectura propuesta, organizada en 5 capas – Infraestructura, Conectividad, Recurso, Recursos y Aplicación – que definen mecanismos básicos que permiten a los usuarios gestionar los recursos compartidos.

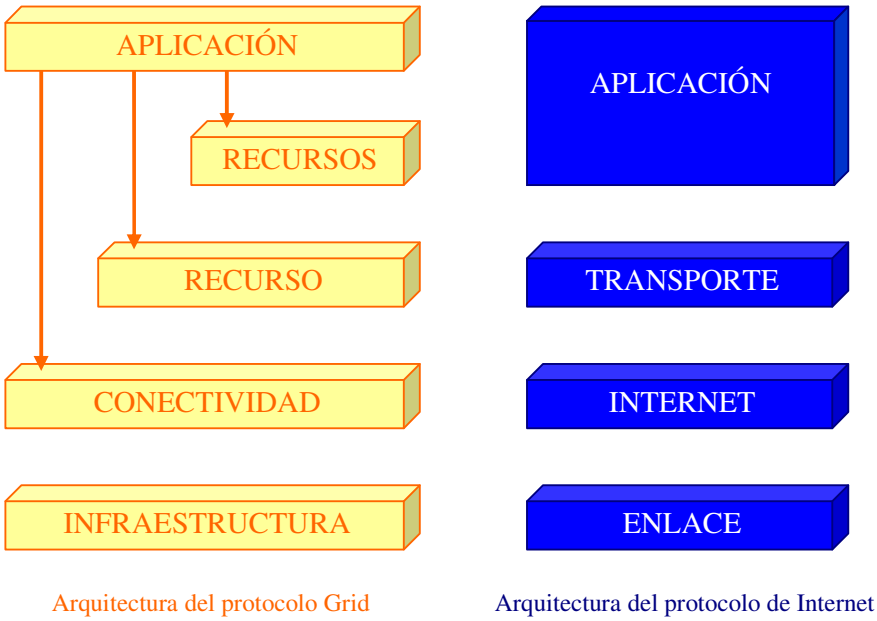


Figura 6.21 Arquitectura de capas de un sistema grid, en relación con la arquitectura de protocolos de internel.

- **Capa de infraestructura (*Fabric*)**→ Según la Figura 6.21, en la primera de las capas, la capa de infraestructura, se encuentran los recursos computacionales que serán compartidos por las organizaciones virtuales (bases de datos, sistemas de almacenamiento de red, clusters) junto con la infraestructura de red y sus mecanismos de gestión y control, donde un recurso puede ser una entidad lógica (como un sistema de archivos distribuido, o un cluster de computadores).

  La funcionalidad de esta capa permite el sharing de operaciones sofisticadas.

  Los recursos específicos presentes en esta capa son:

  - **Recursos computacionales** → Mecanismos requeridos para iniciar programas, controlar y monitorear la ejecución de procesos resultantes. El manejo de mecanismos que permite el control sobre los recursos alojados a procesos son usados como mecanismos avanzados de reserva.

Se necesitan algunas funciones de evaluación para la determinación de características de hardware y software así como información de estado relevante.

- **Recursos alojados** → Se requieren algunos mecanismos para el guardado y recuperación de archivos. Hay mecanismos para la lectura y escritura de subconjuntos de archivos y/o ejecución remota de una selección de datos o reducción de funciones. El manejo de estos mecanismos permiten el control sobre los recursos alojados, para transferencia de datos y otras funciones específicas.
- **Recursos de red** → Mecanismos de manejo que provean control sobre los recursos y la transferencia en la red, funciones que provean las características de red y carga.
- **Repositorios de recursos** → Es una forma especializada en almacenamiento de recursos que requiere mecanismos de manejo de versiones de fuentes y código de objetos.
- **Catálogos** → Forma especializada de almacenamiento que requiere mecanismos para implementar consultas y actualización de operaciones (ej. BD relacionales).

- **Capa de conectividad (*Connectivity*)** → En esta capa se encuentran protocolos estándar de seguridad y comunicación para transacciones de red. Los protocolos de comunicación permiten el intercambio de datos entre la capa más inferior y los recursos mientras que los protocolos de seguridad brindan mecanismos de criptografía para identificar usuarios y recursos.

  Los **Recursos y los protocolos de conectividad** mantienen todas las transacciones específicas de Grid entre diferentes computadores y otros recursos. La red empleada por GRID es Internet, la misma red usada por la web y muchos otros servicios como el correo electrónico. Sin embargo, las tecnologías de Internet direccionan el intercambio de comunicaciones e información entre computadoras pero no provee aproximaciones integradas para el uso coordinado de recursos de diversos sitios de computación.

  Esta capa define el centro de las comunicaciones y autenticación de protocolos requeridos para transacciones específicas de Grid dentro de la red. Los protocolos de comunicación proveen el intercambio de datos entre los recursos de la capa *Fabric*. Los protocolos de autenticación construyen la comunicación de servicios para proveer mecanismos seguros para la verificación e identificación de usuarios y recursos. Las comunicaciones requieren transporte, ruteo, y servicio de nombre (TCP, DNS, IP y ICMP).

  Dentro de los protocolos de capa de conectividad se sitúan algunos protocolos correspondientes a la suite TCP/IP – dado que la comunicación implica por ejemplo ruteo, transporte, se utilizan protocolos IP, ICMP, TCP - así como también el protocolo SSL y certificados X.509 como protocolos estándar que brindan seguridad para acceder a los recursos definidos en la capa de Infraestructura.

  Las soluciones de autenticación para algunos entornos pueden tener características de **Log In**, **Delegación** (de forma que un usuario pueda finalizar

un programa con la habilidad de correr con el provecho de otro usuario y que el programa sea capaz de acceder a recursos para los cuales está autorizado) e **Integración** con varias soluciones de seguridad. Las soluciones de Grid de seguridad deben ser capaces de interoperar con varias soluciones locales. Además, **usuarios basados en las relaciones confiables**: en el cual el sistema de seguridad no debe requerir que cada proveedor de recursos coopere o interactúe con cada una de las configuraciones de los entornos de seguridad.

- **Capa de recurso (*Resource*)** → En el nivel correspondiente a la capa de recurso es donde se encuentran los protocolos que permiten obtener la información de un recurso en particular y gestionarlo controlando el acceso, arranque de procesos, gestión, monitorización y auditoria.

  Las implementaciones de estos protocolos llaman a funciones de capa *Fabric* para acceder y controlar recursos locales. Los protocolos de esta capa se refieren a recursos individuales y por ende ignoran resultados de estado global y acciones atómicas a lo largo de colecciones distribuidas, éstos son de consideración para la capa colección.

  En esta capa se distinguen dos clases principales de protocolos:

- **Protocolo de información** → Para obtener información sobre la estructura y estado de los recursos.
- **Protocolos de manejo** → Usados para negociar el acceso a un recurso compartido, especificando por ejemplo sus requerimientos (incluyendo reservas avanzadas ó calidad de servicio), y operaciones para optimizar estos recursos, como ser la creación de procesos o acceso a datos. A partir de que el manejo de protocolos es responsable para la inicialización de las relaciones del sharing deben servir como puntos de aplicación de política, asegurando que las operaciones requeridas por el protocolo sean consistentes con las políticas bajo las cuales el recurso es compartido. Los protocolos a menudo pueden soportar el monitoreo del estado de una operación y el control de la misma.

- **Capa de recursos o colectiva (*Collective*)** → Al contrario de la capa de recursos (destinada a gestionar un recurso especifico), la siguiente capa definida en la Figura 6.21, denominada *Recursos* o *Colectiva* (*Collective*), contiene los protocolos y servicios que permiten gestionar la interacción de un conjunto de recursos. Algunos ejemplos son los servicios de directorios (que permiten a las organizaciones virtuales descubrir y ubicar recursos compartidos), schedulers distribuidos (que permiten asignar tareas a cada recurso), servicios de monitorización y diagnóstico de recursos ante fallas y servicios de replicación de datos.

  Dado que los componentes de capa colectiva construyen sobre la capa Recursos y la capa Conectividad, pueden implementar una amplia variedad de comportamientos para el sharing sin la localización de nuevos requerimientos sobre los recursos almacenados. Por ejemplo:

  - **Servicios de directorio** → Permite a participantes de las OV descubrir la existencia y/o propiedades de los recursos de OV. Permite a sus usuarios

hacer consultas para recursos por el nombre y/o por atributos, como ser tipo, disponibilidad o carga.

- **Co-Allocation, Scheduling y servicios brokering** → Permiten a recursos de participantes de las OV averiguar sobre más recursos para propósitos específicos y para la programación de tareas sobre los recursos apropiados.
- **Monitoreo y diagnóstico de servicios** → Soportan el monitoreo de los recursos de las OV para fallas, ataques de adversarios, detección de intrusos, sobrecarga y más.
- **Servicios de replicación de datos** → Soporta el manejo de almacenamiento de los recursos pertenecientes a las OV para maximizar el rendimiento en los accesos a datos con las respectivas métricas como el tiempo de respuesta, costo, etc.
- **Sistemas de programación GRID-ENABLED** → Permite usar en los entornos GRID modelos de programación familiar usando varios servicios Grid para localizar recursos, seguridad, etc.
- **Sistemas de ubicación de software** → Descubre y selecciona las mejores implementaciones de software y ejecución de plataformas basadas en parámetros del problema a resolver. Ejemplo: NetSolve y Ninf.
- **Servidor de autorización de la comunidad** → Refuerza el gobierno de las políticas de acceso a recursos generando capacidades que los miembros de la comunidad pueden usar para el acceso a recursos comunitarios.
- **Servicios de cuentas de la comunidad y pagos** → Juntan información acerca del uso de la misma para el propósito de manejo de cuentas, pagos, y/o limitación de usos de recursos a miembros de la comunidad.
- **Servicios de colaboración** → Soportan el intercambio coordinado de información dentro de comunidades de usuarios potencialmente grandes. Mientras los protocolos de capa Recurso deben ser generales en naturaleza y desarrollados ampliamente, los protocolos de capa *Collective* expanden su espectro desde propósitos generales a aplicaciones altas o dominios específicos sólo entre OV específicas. Las funciones de esta capa pueden implementarse como servicios persistentes con protocolos asociados o SDK`s designados para enlazarse con ciertas aplicaciones. En ambos casos sus implementaciones pueden construir en capa Recurso protocolos y API´s.

Los componentes de capa *Collective* pueden crearse para requerimientos de usuarios de comunidades específicas, OV, o dominios de aplicaciones como por ejemplo un SDK que implementa protocolos de coherencia de aplicaciones específicas, o un servicio co-reservado para un conjunto específico de recursos en la red. Otros componentes de esta capa pueden ser de propósitos más generales por ejemplo, servicios de replicación que manejen una colección internacional de sistemas de almacenamiento para múltiples comunidades o un servicio directorio designado a permitir el descubrimiento de OV´s.

Los **servicios colectivos** (*collective services*) se basan en protocolos: protocolos de información que obtienen datos sobre la estructura y estado de los recursos, y protocolos de manejo que negocian el acceso a recursos de una forma uniforme.

- **Capa de aplicación (*Application*)** → La última capa definida es denominada Aplicación (*Application*). En esta se encuentran definidos los protocolos que permiten acceso a la estructura Grid. Las aplicaciones son construidas en términos de servicios definidos para alguna de las capas antes mencionadas, pudiendo, por ejemplo, comunicarse directamente con una capa en particular. Cada una de estas capas tiene protocolos bien definidos que proveen acceso al uso de servicios: manejo de recursos, acceso a datos, y más.

  Si se considera que una aplicación de usuario necesita analizar datos contenidos en archivos independientes, tendrá que realizar entonces las siguientes tareas básicas:

  - Obtener la credencial necesaria de autenticación para abrir los archivos (recursos y protocolos de conectividad).
  - Consultar el sistema de información y réplica de catálogos para determinar dónde pueden encontrarse las copias de los archivos en GRID así como también dónde se hallan los recursos más convenientes para hacer el análisis de datos (***Collective Services***).
  - Pasar los pedidos a la capa de **Infraestructura**, la computadora apropiada, sistema de almacenamiento y redes, para extraer los datos, iniciar los procesos, y proveer los resultados (recursos y protocolos de conectividad).
  - Monitorear el progreso de varios procesos y transferencia de datos, notificando al usuario cuándo el análisis se completa y también detectando y respondiendo ante situaciones de fallas (***collective services***).

  Hoy día hay diversos tipos de redes disponibles, las cuales se caracterizan por su tamaño (locales, nacionales e internacionales) y según su rendimiento en términos de throughput (cantidad de datos transferidos desde un lugar a otro en un tiempo determinado). Típicamente el throughput se mide en Kbps, Mbps or Gbps. Grid está construida sobre redes de alto rendimiento, tal es el caso de la red Intra-Europe GEANT o la red de UK Super Janet la cual exhibe 10 Gbsp de rendimiento en el *backbone* de red (*backbone*: se utiliza para nombrar a aquellos enlaces de alta velocidad en las redes que unen grandes nodos).

  A medida que crece la velocidad de la red, el poder de GRID es determinado por el rendimiento de los recursos de computación disponibles en los nodos de la red. Los nodos mayores serán recursos de alto rendimiento como un cluster largo de computadoras o cualquier supercomputador dedicado.

  Como ejemplo en la Figura 6.22 se propone una lista parcial de componentes correspondientes a cada una de las capas antes descritas, que podrían ser utilizados en un sistema grid relacionado a administración de réplicas.

FUNCIONAMIENTO DE LA COMPUTACIÓN GRID

Uno de los principales problemas que trae aparejada la implementación de la computación grid es la forma en que se administrarán los recursos que son parte de su infraestructura.

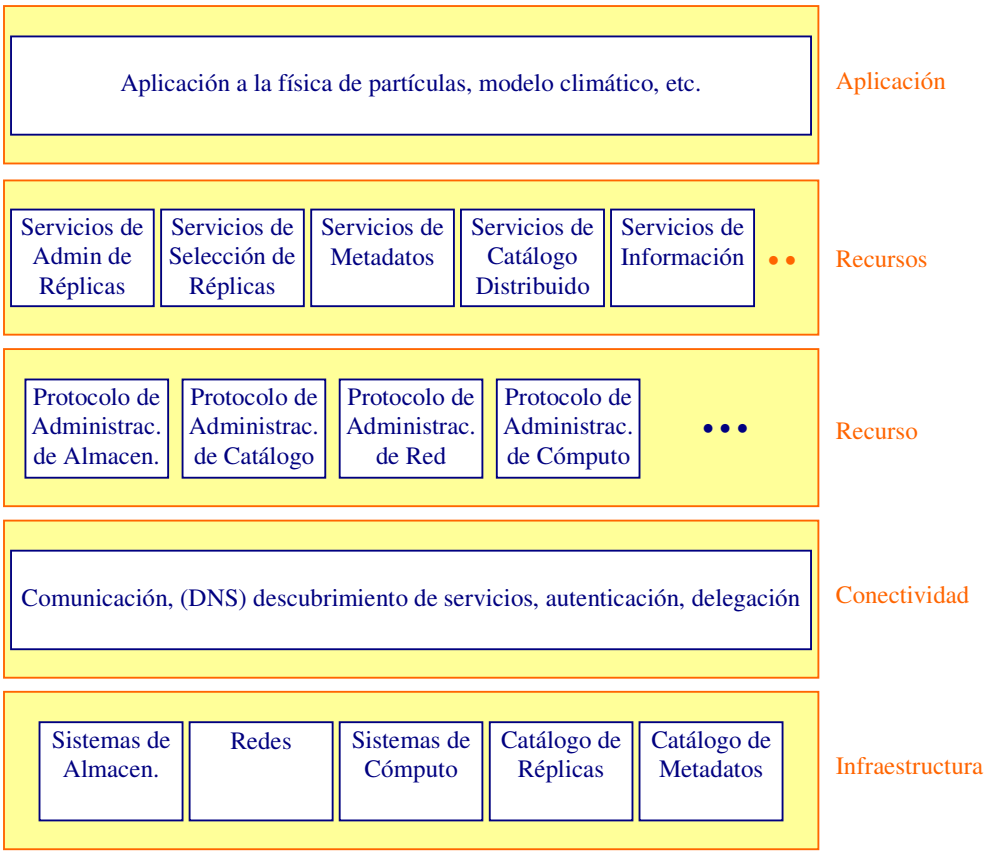


Figura 6.22 Lista parcial de componentes que se pueden encontrar en cada capa.

Los recursos que utiliza la computación grid se dividen en:

- Recursos que forman parte de la infraestructura, tales como recursos de hardware, de software y de red.
- Recursos que forman parte de los requerimientos; por ejemplo, el *tiempo* que se dispone de los recursos de infraestructura, los *datos* requeridos para un proceso dado, etc.

Por ser un sistema de cómputo distribuido, la computación grid presenta los problemas típicos de éstos, por ejemplo a la hora de acceder a los recursos de cómputos y a los de red. Como se trata de un sistema dinámico, los recursos cambian y, por lo tanto, se deben utilizar sistemas que se encarguen de buscar recursos disponibles. De la misma forma en que se pueden agregar nodos que aporten recursos, más tarde pueden desaparecer, reduciéndose así los recursos que aportaban y el software que pudieran tener. También puede haber pérdida de recursos por una falla en las comunicaciones.

Para optimizar las infraestructuras distribuidas que forman el grid y facilitar su utilización, surge la necesidad de una arquitectura global que se encargue de prestar los servicios de asignación y planificación de recursos. Un ejemplo de esto es el proyecto CrossGrid que está orientado a aplicaciones paralelas e interactivas.

Este proyecto grid se centra en la utilización de un *middleware* que tiene la tarea de planificar y seleccionar eficientemente los recursos. A tal efecto cuenta con un *Resource Broker* como entidad principal, que presta servicios para la ejecución de trabajos secuenciales. Cuando un usuario envíe un trabajo éste será el encargado de buscar y seleccionar los recursos necesarios.

Para dar respaldo en el control de aplicaciones paralelas interactivas se cuenta con extensiones de los servicios prestados por el *Resource Broker*. Una de estas extensiones es el *Scheduling Agent*, una extensión de la funcionalidad que proporciona el *Resource Broker* como punto de acceso al Grid para el usuario, y capaz gestionar los recursos y hacer que los procesos se ejecuten de manera transparente.

En esta arquitectura, los recursos disponibles se identifican como nodos de computación. Estos nodos representan una granja local compuesta por diferentes nodos de trabajo. También existen nodos de almacenamiento desde donde se tendrá acceso a los datos.

Cuando se debe realizar un trabajo, el encargado de verificar qué recursos están disponibles es el *Resource Selector*. La asignación de dichos recursos dependerá si se trata de un trabajo secuencial o paralelo. Si se trata de un trabajo secuencial se deberá ejecutar en una única granja y requerirá un solo nodo de trabajo o CPU. A tal efecto existirá una lista de posibles granjas que podrían realizar el trabajo, estando ordenadas según algunas preferencias (tiempo ocioso de CPU, memoria, etc.). Un trabajo que requiera procesamiento paralelo necesitará varias CPUs y habrá que tener en cuenta que primero se tratarán de seleccionar granjas con todos sus procesadores libres, para evitar tener que establecer una comunicación entre clusters y sufrir la latencia producida por sus comunicaciones; luego, se intentará formar grupos de granjas que reúnan el número de CPUs libres requeridas. Y, por último, se tendrá un ordenamiento de grupos de menor a mayor cantidad de granjas diferentes y, para los que igualen en elementos, se utilizará una función de ponderación de sus componentes.

Un problema que surge de la utilización de estas listas de recursos es que, puede ocurrir que no se encuentren actualizadas completamente en tiempo real, y puede darse el caso en que el *Resource Broker* o el *Scheduling Agent* considere una CPU como libre cuando realmente esté ocupada. Para evitar esto se trabaja con un método de reservas locales temporales. Una vez decidido a qué grupo se asignarán esos recursos, se les reserva por un espacio de tiempo, y no serán tenidos en cuenta para trabajos siguientes.

También se utiliza el concepto de prioridades, que permite que una aplicación se ejecute antes que otra. En el caso en que todos los recursos estén ocupados, se podrá decidir qué tarea será suspendida temporalmente para dar paso a otra de mayor prioridad.

Una vez realizadas todas las consideraciones sobre los recursos y prioridades se debe permitir la ejecución del trabajo. El encargado de enviar los ejecutables de forma confiable es el *Application Launcher*, que hará la distribución a todos los recursos seleccionados que se encuentren implicados en la realización del trabajo.

Hasta aquí se ha dado una aproximación de cómo interactúan los distintos componentes del grid para que un paquete de procesamiento enviado por un usuario se resuelva de la forma más eficiente y transparente posible.

VENTAJAS Y DESVENTAJAS DE LA COMPUTACIÓN GRID

La Computación Grid se ha creado con el fin de brindar una solución a determinadas cuestiones, como problemas que requieren de un gran número de ciclos de procesamiento, o acceso a una gran cantidad de datos. Encontrar un hardware y un software que permitan brindar estas utilidades comúnmente proporciona inconvenientes de costes, seguridad y disponibilidad. En ese sentido, se integran diferentes tipos de máquinas y de recursos; por lo tanto, una red grid nunca queda obsoleta, todos los recursos se aprovechan (si se renuevan todos los PCs de una oficina, se pueden incorporar los antiguos y los nuevos).

Por otra parte, esta tecnología brinda a las empresas el beneficio de la velocidad, lo que supone una ventaja competitiva, con lo cual se provee una mejora de los tiempos para la producción de nuevos productos y servicios.

Facilita la posibilidad de compartir, acceder y gestionar información, mediante la colaboración y la flexibilidad operacional, aunando no sólo recursos tecnológicos dispares, sino también personas y aptitudes diversas. Otro de los aspectos al que se tiende es a incrementar la productividad otorgando a los usuarios finales acceso a los recursos de computación, datos y almacenamiento que necesiten, y cuando los necesiten.

Con respecto a la seguridad en la grid, ésta está sustentada con las “intergrids”, donde esa seguridad es la misma que ofrece la red Lan sobre la cual se utiliza tecnología grid.

El paralelismo puede estar visto como un problema, por ser una máquina paralela muy costosa. Pero, si tenemos disponibilidad de un conjunto de máquinas heterogéneas de pequeño o mediano tamaño cuya potencia computacional sumada sea considerable, permitiría generar sistemas distribuidos de muy bajo coste y gran potencia computacional.

Grid computing necesita, para mantener su estructura, de diferentes servicios como Internet, conexiones de 24 horas, los 365 días, con banda ancha, servidores de capacidad, seguridad informática, VPN, firewalls, encriptación, comunicaciones seguras, políticas de seguridad, normas ISO, y algunas características más… Sin todas estas funciones y características no es posible hablar de *Grid Computing*.

La tolerancia a fallos significa que si una de las máquinas que forman parte del grid colapsa, el sistema lo reconoce y la tarea se reenvía a otra máquina, con lo cual se cumple el objetivo de crear infraestructuras operativas flexibles y resistentes.

APLICACIONES DE LA COMPUTACIÓN GRID

Existen cinco aplicaciones generales para la computación grid:

- **Súper computación distribuida** → Son aquellas aplicaciones cuyas necesidades no pueden ser satisfechas en un único nodo. Las necesidades se producen en instantes de tiempo determinados y consumen muchos recursos.
- **Sistemas distribuidos en tiempo real** → Son aplicaciones que generan un flujo de datos a alta velocidad que debe ser analizado y procesado en tiempo real.
- **Servicios puntuales** → Aquí no se tiene en cuenta la potencia de cálculo y capacidad de almacenamiento sino los recursos que una organización puede considerar como no necesarios. Grid presenta a la organización esos recursos.
- **Proceso intensivo de datos** → Son aquellas aplicaciones que hacen un gran uso del espacio de almacenamiento. Este tipo de aplicaciones desbordan la capacidad de almacenamiento de un único nodo y los datos son distribuidos por todo el grid. Además de los beneficios por el incremento de espacio, la distribución de los datos a lo largo del grid permite el acceso a los mismos de forma distribuida.
- **Entornos virtuales de colaboración** → Área asociada al concepto de Teleinmersión, de manera que se utilizan los enormes recursos computacionales del grid y su naturaleza distribuida para generar entornos virtuales 3D distribuidos.

Existen aplicaciones reales que hacen uso de mini-grids, las cuales están centradas en el campo de la investigación en el terreno de las ciencias físicas, médicas y del tratamiento de la información. Además existen diversas aplicaciones en el campo de la seguridad vial; por ejemplo, este sistema permite traducir el riesgo de herir a un peatón y la resistencia del parachoques de un vehículo en una serie de datos que ayudan a diseñar la solución de protección más adecuada.

Entre los primeros proyectos *grid*, surge *Information Power Grid (IPG)*, que permite la integración y gestión de recursos de los centros de la NASA. El proyecto SETI@Home a nivel mundial, de investigación de vida extra-terrestre, o búsqueda de vida inteligente en el espacio, puede ser considerado como precursor de esta tecnología, si bien la idea de Grid Computing es mucho más ambiciosa puesto que no sólo se trata de compartir ciclos de CPU para realizar cálculos complejos sino que se busca la creación de una infraestructura de computación distribuida, con interconexión de diferentes redes, de definición de estándares, de desarrollo de procedimientos para la construcción de aplicaciones, etc.

PROYECTOS EN DESARROLLO BAJO EL SISTEMA DATA GRID

- **Globus Project** → El Proyecto Globus es una iniciativa multi-institucional para la investigación y el desarrollo de tecnologías fundamentales para Grids, con la activa participación de la empresa IBM, cuya intención principal es crear una plataforma completa donde compartir aplicaciones y recursos informáticos en Internet. Uno de los primeros productos desarrollados por el Proyecto Globus es el Globus Toolkit, que está siendo utilizado en varios proyectos de aplicación y despliegue de Grid en los Estados Unidos, Europa y el resto del mundo. El Proyecto Globus tiene su sede central en el Laboratorio Nacional Argonne y la Universidad del Instituto de Ciencias de Información de California del Sur.

  El proyecto permite llevar las redes Grid más allá de las habituales aplicaciones técnicas y científicas para que pueda ser de utilidad en aplicaciones reales de

negocio, conectando muchos superordenadores dispersos geográficamente mediante Internet y unos protocolos específicos de código abierto creados por la organización internacional Globus (globus.org).

La Arquitectura de Servicios Abiertos Grid (OGSA) presenta un conjunto de especificaciones y estándares que combina los beneficios de la informática Grid y los servicios web. Así, los clientes pueden, por primera vez, compartir y acceder a los recursos informáticos que necesitan en Internet, contando con el soporte de una infraestructura muy resistente, con capacidad de autogestión y siempre disponible; pueden integrar aplicaciones y compartir datos y potencia de procesado, consiguiendo unos niveles de eficiencia muy altos, así como muy bajos costes.

Este conjunto de especificaciones OGSA completa los estándares XML, WDSL y SOAP (todos ellos importantes para los servicios web), con los estándares desarrollados por Globus para tecnologías de redes Grid, utilizados para localizar, planificar y asegurar recursos informáticos.

OGSA cuenta con el apoyo de empresas de diferentes industrias, incluyendo AVAKI, proveedor de soluciones comerciales de software Grid; Entropía, proveedor de informática de redes Grid distribuida basada en PC; Microsoft; y Platform Computing, proveedor de software de informática distribuida.

IBM tiene como objetivo la implantación de OGSA como punto clave en su "Proyecto eLiza". El proyecto eLiza es la iniciativa de informática autonómica de IBM para construir un servidor de infraestructura autogestionable, abierto y heterogéneo para el comercio electrónico y la puesta en práctica de Grids comerciales.

- **TeraGrid** → Es un proyecto estadounidense, llevado adelante por la Fundación Nacional de Ciencias (NSF). Dedicado a la investigación científica abierta, tiene el objetivo de interconectar instalaciones y centros de investigación académica en puntos distantes geográficamente, está considerado como una de las infraestructuras más grandes y más rápidas del mundo.

  A fin de permitir que los investigadores lleven a cabo proyectos de investigación y colaboren en ellos, se archivan datos de varias disciplinas científicas y de ingeniería. Los proyectos de esta magnitud son de valor crítico y fundamental para la comunidad científica y la recompensa que surja de la integración de soluciones para los investigadores a esta escala podría ser enorme, al reducir los costes y el tiempo de desarrollo, y al crear productos más seguros.

  Mediante el proyecto TeraGrid, los científicos tendrán la capacidad para simular actividades sísmicas en estructuras a fin de diseñar edificios y puentes más seguros, los astrónomos podrán compartir datos desde sus telescopios y los investigadores médicos tendrán la posibilidad de compartir ideas y datos para curar quizás una enfermedad, lo que asegura que las oportunidades que se ofrecen a los científicos son incalculables

- **CrossGrid** → Con el fin de poder ejecutar aplicaciones interactivas en un entorno Grid, se modifica el *middleware*, adaptándolo para ello. En este proyecto se han definido cuatro aplicaciones que utilizarán desarrollos Grid comunes, que son: 1) Simulación interactiva y visualización de un sistema biomédico; 2) Sistema de apoyo a un equipo de crisis por inundaciones; 3) Análisis de datos distribuidos en Física de Altas Energías y 4) Previsión meteorológica y modelización de la contaminación atmosférica.

  Los proyectos científicos relacionados con áreas como el análisis de la física de partículas, la biología computacional, la medicina, las ciencias medioambientales y la astrofísica, se desarrollan en el Reino Unido bajo el nombre de e-Ciencia, la cual sólo puede ser comprendida con el avance de la tecnología Grid o de la computación distribuida. Precisamente, en torno a esta tecnología y a la e-Ciencia han surgido numerosos proyectos, y múltiples centros de investigación se han centrado en su desarrollo.

  Por su parte, CrossGrid, también proyecto de la Unión Europea nacido en 2001, ha tenido como fin el desarrollo, entre los años 2002 y 2005, de aplicaciones interactivas en entornos Grid y la extensión del banco de ensayo (testbed) de DataGrid.

- **OpenMoIGRID** → Desarrollado por ComGenex Inc., un proveedor de descubrimientos químicos avanzados, tiende a proporcionar información unificada y extensible del entorno para solucionar las cuestiones moleculares de diseño e ingeniería con relación a la química, la farmacia y la bioinformática. El proyecto utiliza un set de aplicación con herramientas orientadas a establecer servicios centrales Grid y funciones provistas por la infraestructura Eurogrid, y se desarrollan herramientas para aumentar el acceso a las bases de datos heterogéneas y de distribución y para adaptar las herramientas de software existentes.

  ComGenex es un proveedor de soluciones químicas integrales para las industrias farmacéutica y biotecnológica. Esta compañía dedicada a la formación de compuestos químicos, facilita el desarrollo de medicamentos en Europa. Cuenta con varias tecnologías propias de síntesis y análisis en las áreas de la química, la producción de instrumentos, la optimización de derivados, el análisis químico con aplicación médica y la bioinformática, y es compatible con ComGenexDirect, el primer sistema de comercio electrónico de la industria en el campo de los descubrimientos de nuevos fármacos.

  ComGenex, el único socio comercial entre los cinco grupos de investigación en el proyecto, proporciona diseño químico, química de altas prestaciones, biología celular y molecular, desarrollo de ensayos, monitorización de altas prestaciones y experiencia química y bioinformática.

- **UK e-Science** → Un equipo de científicos del Reino Unido dio a conocer en una conferencia de gran magnitud, un elemento clave de la computación Grid que facilitará a los investigadores aprovechar enormes recursos informáticos de todo el mundo para afrontar los desafíos científicos clave en campos como el genoma humano y la física de partículas. Los responsables de la iniciativa elaboraron un

conjunto de procedimientos que permitirán que los científicos que utilicen Grid accedan a las bases de datos de los resultados de investigación procedentes de sistemas que se encuentren en cualquier parte del mundo.

La intención es que las nuevas especificaciones sirvan para crear sistemas de prototipos de Grid, así como aplicaciones comerciales y científicas.

- **EGEE (Enabling Grids for e-Science in Europe)** → El proyecto utiliza la tecnología Grid para interconectar recursos computacionales de veintisiete países europeos, con el objeto primordial de unir los recursos de los equipos informáticos de las instituciones participantes y crear de este modo un supercomputador virtual, aprovechando la infraestructura de comunicación de banda ancha proporcionados por la Red Europea de Investigación Géant.

  En el proyecto se plantea coordinar el uso conjunto de recursos de computación en forma distribuida entre los diferentes centros, constituyendo una plataforma donde aplicaciones de gran importancia en Informática Biomédica y en Física de Altas Energías funcionarán a máximo rendimiento. El centro que dirige el proyecto es el Laboratorio Europeo de Física de Partículas (CERN) que se encuentra en Ginebra (Suiza).

  El Grupo de Redes y Computación de Altas Prestaciones de Valencia (GRyCAP) ha coordinado en los últimos años unos diez proyectos, la mayor parte de ellos europeos, orientados al desarrollo de aplicaciones informáticas avanzadas para el diagnóstico por imagen o la planificación quirúrgica. El GRyCAP se encarga en el proyecto EGEE de identificar, seleccionar y apoyar el despliegue de aplicaciones informáticas de apoyo a la asistencia sanitaria de forma que los usuarios médicos puedan tener acceso a las más avanzadas herramientas para el diagnóstico, la prevención o la simulación de la terapia tanto para la investigación clínica como la práctica diaria.

  Asimismo, cuenta con una avanzada infraestructura informática formada por un Grupo de computadores para el Grid y con facilidades de almacenamiento de datos. El objetivo es participar en la puesta a punto de un sistema computacional internacional montado en Grid que cubra las necesidades informáticas generadas por el procesado de datos procedentes de los experimentos de física de partículas. Esta iniciativa consiste en aglutinar los recursos informáticos con el fin de ser compartidos por investigadores de diferentes áreas científicas como la Astrofísica, la Meteorología, la Biología, la Nanofotónica, etc