1

CONCEPTOS GENERALES

1.1. Necesidad del paralelismo

En la actualidad, hay una gran demanda de computadores rápidos para muchos campos de aplicación de tipo científico, técnico, médico, etc. Para muchas de estas áreas de aplicación, se necesita un gran volumen de cálculo. Sin el empleo de computadores muy potentes, muchos de los cálculos precisos para este tipo de aplicaciones no podrían realizarse a una velocidad razonable. Por otra parte, las tendencias actuales de la ciencia y la técnica siguen planteando problemas que exigen cada vez más y más potencia de cálculo. Todo esto hace que las prestaciones de los computadores deban mejorar continuamente.

Para ver de una forma más palpable la necesidad actual de computadores más rápidos pondremos un ejemplo: los satélites artificiales envían información a la tierra al ritmo de 10^{20} bits/sg. Todo este volumen de información se utiliza para efectuar predicciones meteorológicas, etc. Si queremos procesar esta información con el fin de que los resultados de ese proceso sean útiles, deben procesarse con una velocidad suficiente. Teniendo en cuenta que los cálculos para este tipo de aplicaciones no son sencillos, se podría estimar que la velocidad de proceso necesaria, para que los resultados se obtengan antes de dejar de ser útiles, debe ser de unas 10^{23} operaciones/sg.

Es conocido que la velocidad de los computadores ha ido aumentando con el tiempo según las evoluciones de la Tecnología. En cualquier caso, existen **límites físicos** para el aumento indefinido de la velocidad de las máquinas (velocidad de la luz, capacidad de los conductores, disipación de calor en los circuitos integrados, etc.). Estas limitaciones provocan que se busquen nuevas alternativas para aumentar las prestaciones de las máquinas. Una de las alternativas que resulta más evidente es poner a trabajar varios elementos de proceso simultáneamente, dicho de otra forma, tratar de desarrollar el **paralelismo**; así el trabajo se repartirá y se tardará menos tiempo en completar el proceso de cálculo.

1.2. Concepto de paralelismo

Definiremos el **procesamiento paralelo** como *la forma de computación que explota la concurrencia en el proceso de cálculo*.

En esta definición, la palabra **concurrencia** se refiere a la *ejecución simultánea de diferentes acciones*.

El procesamiento paralelo puede explotarse a diferentes niveles (Gajski & Peir, 1985). Describiremos estos niveles en orden descendente:

Nivel de trabajo (*job*): se trata de ejecutar simultáneamente diferentes trabajos (procesos) independientes en diversos elementos de proceso. Este nivel de paralelismo lo puede manejar completamente el gestor de procesos del sistema operativo (*scheduler*), asignando cada trabajo a un procesador diferente. En este nivel una de las explotaciones más sencillas del paralelismo es el **multiproceso simétrico** (SMP) que consiste en tener varias colas de procesos, asignando cada una a un procesador diferente.

Nivel de procedimiento: este nivel trataría de ejecutar diferentes procedimientos de un mismo proceso de forma simultánea sobre elementos de proceso diferentes.

Nivel de instrucción: se trataría de ejecutar diferentes instrucciones, de una misma tarea simultáneamente. Una de las formas más fáciles de explotar el paralelismo a este nivel, es provocar que iteraciones diferentes de un bucle sean ejecutadas en procesadores distintos.

Nivel intrainstrucción: explota el paralelismo dentro de cada instrucción, ejecutando de forma simultánea varias acciones de cara a la ejecución de las instrucciones. Un ejemplo de este nivel de paralelismo lo constituyen los procesadores segmentados. En ellos se divide la ejecución de cada instrucción en pasos, y cada paso es ejecutado por una etapa diferente del procesador (segmento), de forma que pueden ejecutarse simultáneamente acciones diferentes de distintas instrucciones.

Es posible implementar el paralelismo a uno o más de estos niveles (incluso a todos) en un sistema computador. En el nivel más alto, el paralelismo puede llevarse a cabo por software, y por tanto, más fácilmente; por el contrario, en el más bajo, el paralelismo sólo puede efectuarse dentro de la propia CPU. En los niveles intermedios puede haber participación tanto del hardware como del software, predominando éste en los niveles más altos.

1.3. Limitaciones del paralelismo

Como veremos a continuación, el paralelismo tiene sus limitaciones. Una de ellas radica en el hecho de que los procesos suelen tener una parte que no es paralelizable. También se debe tener en cuenta que la paralelización de un proceso requiere tanto software adicional, como comunicaciones entre procesadores. Ambos factores suponen pérdidas de tiempo añadidas.

1.3.1. Rendimiento de los computadores paralelos

La velocidad con que operan los computadores se mide por el número de operaciones básicas que pueden realizar por unidad de tiempo. Una unidad para medir esta velocidad es el **MIPS** (1 millón de instrucciones/sg.), sin embargo esta unidad tiene el inconveniente de que no es homogénea porque las instrucciones de un procesador pueden ser mucho más potentes que las de otro. Una unidad algo más fiable es el **MFLOPS** (1 millón de instrucciones de punto flotante/sg.) que, aunque en menor medida, también adolece del mismo defecto. Otra unidad que se emplea mucho es el **VUP** (*VAX unit performance*) que es la velocidad del computador en cuestión tomando como unidad la del VAX-11/780. Actualmente para este propósito se utilizan unos programas de prueba específicos denominados *benchmarks*; el tiempo de ejecución de estos programas nos dará una medida del rendimiento de un sistema. Para medida de la CPU el que se utiliza actualmente es el SPEC CPU2006.

Una medida bastante utilizada para medir la eficiencia de un sistema, y que sólo afecta a parámetros arquitectónicos del mismo sin entrar en la calidad de la tecnología empleada, es el ***CPI*** (**ciclos por instrucción**) que es el número medio de ciclos de reloj necesario para ejecutar una instrucción. Dicho de otra manera:

$$CPI = \frac{N^o \, de \, ciclos \, de \, reloj \, consumidos}{N^o \, de \, instrucciones \, ejecutadas} \qquad [1.1]$$

Si un programa tiene n instrucciones, su tiempo de ejecución vendrá dado por:

$$t = CPI * n * \frac{1}{f} \qquad [1.2]$$

donde f es la frecuencia de reloj, y, por tanto, su inverso es el tiempo de ciclo del reloj.

Una forma de medir la calidad de un sistema paralelizado consiste en comparar la velocidad conseguida con el sistema paralelo (con N procesadores) con la velocidad conseguida con un solo procesador. Por ello, definiremos la **ganancia de velocidad** o **aceleración** (***speed up***) de un sistema de N procesadores como

$$S(N) = \frac{t(1)}{t(N)} \qquad [1.3]$$

donde $t(1)$ es el tiempo empleado para ejecutar el proceso si se ejecuta en un solo procesador y $t(N)$ es el tiempo empleado para ejecutarlo en el sistema paralelo con N procesadores. En condiciones ideales, $t(N) = 1/N$ con lo que, en esas mismas condiciones, la ganancia de velocidad dada por la ecuación 1.3 será

$$S(N)_{ideal} = \frac{t(1)}{t(N)} = \frac{1}{1/N} = N \qquad [1.4]$$

Una forma de medir el rendimiento del sistema, será comparar la ganancia de velocidad del sistema con la ganancia de velocidad ideal dada por 1.4, a esta medida se la denomina **eficiencia** y vendrá dada por:

$$E(N) = \frac{S(N)}{S(N)_{ideal}} = \frac{S(N)}{N} = \frac{t(1)/t(N)}{N} = \frac{t(1)}{Nt(N)} \qquad [1.5]$$

que indica la medida en que se aprovechan los recursos.

1.3.2. Ley de Amdahl

La ley de Amdahl (1967) pone un límite superior a la ganancia en velocidad, y por tanto también a la eficiencia, de un sistema paralelo atendiendo al hecho, anteriormente comentado, de que los procesos suelen tener partes que no pueden ser ejecutadas en paralelo, sino sólo de forma secuencial pura.

Del tiempo total de ejecución del proceso, llamaremos s a la parte que no puede ser paralelizada y p al resto (paralelizable). Suponiendo condiciones ideales en la parte paralelizable del proceso p, tendremos que los tiempos mínimos de ejecución del proceso con un solo procesador y en el sistema paralelo con N procesadores serán

$$\begin{aligned} t(1) &= s + p \\ t(N) &= s + \tfrac{p}{N} \end{aligned} \qquad [1.6]$$

y, por tanto, la ganancia de velocidad máxima vendrá dada por:

$$S(N)_{máx} = \frac{s+p}{s+\frac{p}{N}} = \frac{1}{\frac{s}{s+p} + \frac{p}{N(s+p)}} \qquad [1.7]$$

Llamando f a la fracción de tiempo no paralelizable sobre el total dada por

$$f = \frac{s}{s+p} \qquad [1.8]$$

tendremos que 1.7 se transforma en

$$S(N)_{máx} = \frac{1}{f + \frac{1-f}{N}} = \frac{N}{Nf + 1 - f} = \frac{N}{1 + (N-1)f} \qquad [1.9]$$

Puede verse que, si el número de procesadores crece indefinidamente, la ganancia máxima tenderá a

$$S_{lím} = \lim_{N\to\infty} \frac{N}{1+(N-1)f} = \frac{1}{f} \qquad [1.10]$$

Debido a esta limitación de la ganancia máxima, la fracción no paralelizable f recibe el nombre de **cuello de botella secuencial** de un proceso.

Atendiendo a la definición de eficiencia dada por 1.5, y teniendo en cuenta lo anterior, la eficiencia máxima será:

$$E(N)_{máx} = \frac{S(N)_{máx}}{N} = \frac{1}{1+(N-1)f} \qquad [1.11]$$

Si el número de procesadores crece indefinidamente, la eficiencia máxima tenderá a

$$E_{lím} = \lim_{N\to\infty} \frac{1}{1+(N-1)f} = 0$$

Estas cotas superiores constituyen la ley de Amdahl. Ilustraremos los efectos de esta ley con un pequeño ejemplo:

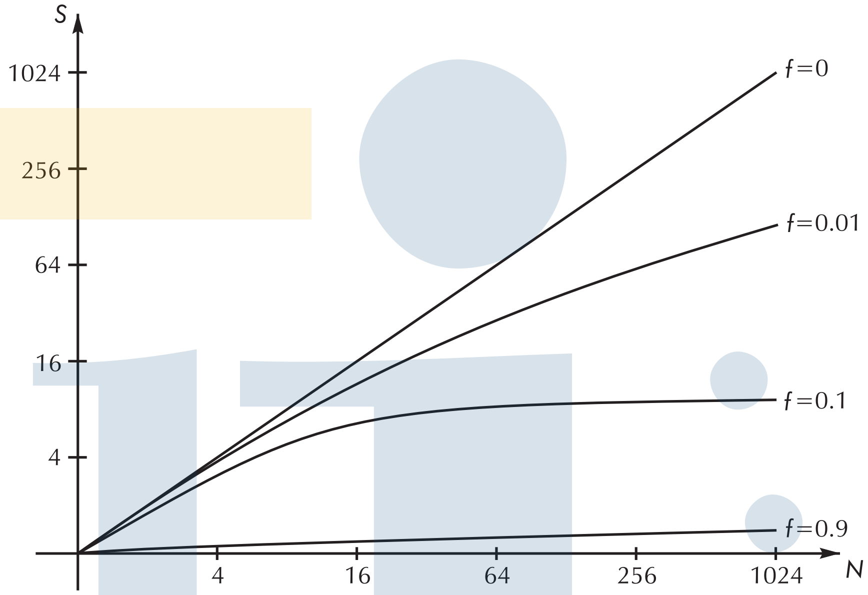

Fig. 1.1. Variación de la ganancia de velocidad en función del número de procesadores para diversos valores de f.

Ejemplo 1.1: *Veamos como disminuye el rendimiento de un sistema con 16 procesadores cuando sobre él se ejecuta un proceso con un 25 % no paralelizable. Según la expuesta ley de Amdahl, la ganancia máxima de velocidad vendrá dada por:*

$$S(16)_{máx} = \frac{N}{1 + (N-1)f} = \frac{16}{1 + 0{,}25 * 15} = 3{,}37$$

y la eficiencia estará acotada por

$$E(16)_{máx} = \frac{1}{1 + (N-1)f} = \frac{1}{1 + 15 * 0{,}25} = 0{,}21$$

Como se ha podido ver mediante este pequeño ejemplo, una pequeña parte no paralelizable en un proceso puede provocar una fuerte disminución del rendimiento del sistema.

Por lo dicho hasta ahora, la ley de Amdahl es bastante pesimista. Es necesario señalar, sin embargo, que esta ley no tiene en cuenta algunos beneficios de los sistemas paralelos. Por ejemplo, un sistema con varios procesadores que tengan memorias caché propias puede hacer que cada procesador almacene en su caché los datos que está utilizando; sin embargo, un procesador solo, con una memoria caché del mismo tamaño, no podrá hacerlo con todos los datos. Esto puede causar que, en estas condiciones, la velocidad de un sistema paralelo con N procesadores pueda aumentar por encima de N, lo que aumentaría la cota superior propuesta por la ley de Amdahl. Una realidad que sí transmite la ley de Amdahl es que **el rendimiento no aumenta por incrementarse indefinidamente el número de procesadores**.

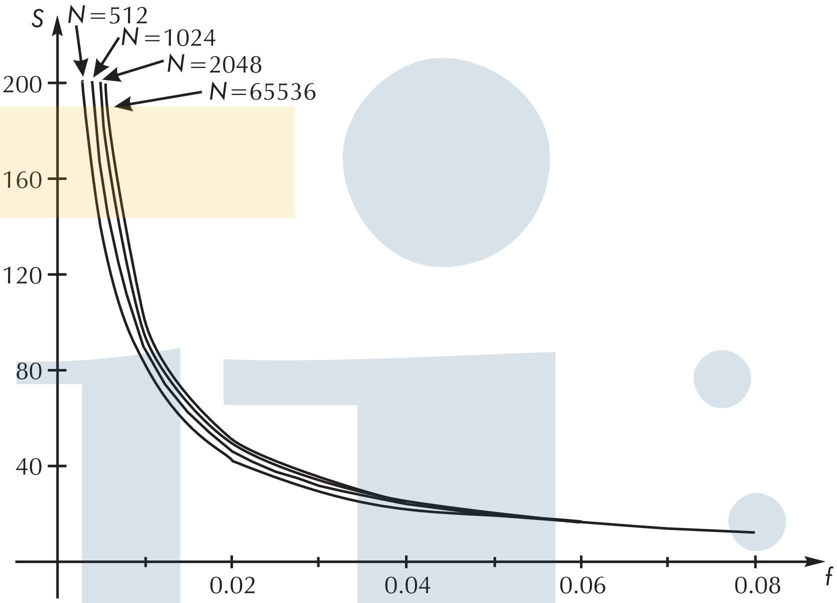

Fig. 1.2. Variación de la ganancia de velocidad en función del cuello de botella secuencial (f) parametrizado según el número de procesadores.

En la figura 1.1 se muestra la variación de la ganancia máxima de velocidad en función del número de procesadores para diferentes valores del cuello de botella secuencial (f). Nótese que las escalas de esta gráfica son logarítmicas. Por otra parte, en la figura 1.2 se muestra la variación de la ganancia máxima de velocidad en función de la fracción de proceso no paralelizable para sistemas con diferente número de procesadores. Como puede verse en esta figura, cuando la fracción secuencial empieza a aumentar la ganancia es prácticamente independiente del número de procesadores empleados.

1.3.3. Ley de Gustafson

La ley de Gustafson (1988) viene a compensar el pesimismo dejado por la ley de Amdahl. Ésta se refería a problemas con volumen de cálculo fijo en que se aumenta el número de procesadores. Sin embargo, en la práctica, el volumen del problema no es independiente del número de procesadores, ya que con mayor número de procesadores se pueden abordar problemas de mayores dimensiones. Por ello, la ley de Gustafson se refiere al crecimiento del volumen de cálculo necesario para resolver un problema. Como veremos a continuación, en la mayoría de los casos, cuando el volumen del problema crece, lo hace sólo en su parte paralela, no en su parte secuencial. Ello hace que el cuello de botella secuencial tienda a cero cuando el volumen del problema aumenta.

La razón de que cierto tipo de problemas adquieran gran volumen de cálculo es la disminución del tamaño de la malla de cálculo o también el aumento la extensión espacio-temporal del problema. Esto hace que el número de puntos aumente de forma cúbica respecto al grado de disminución en la malla, si el problema es tridimensional. Hay muchos problemas en que, además de las tres dimensiones del espacio, también interviene el tiempo, por lo que el aumen-

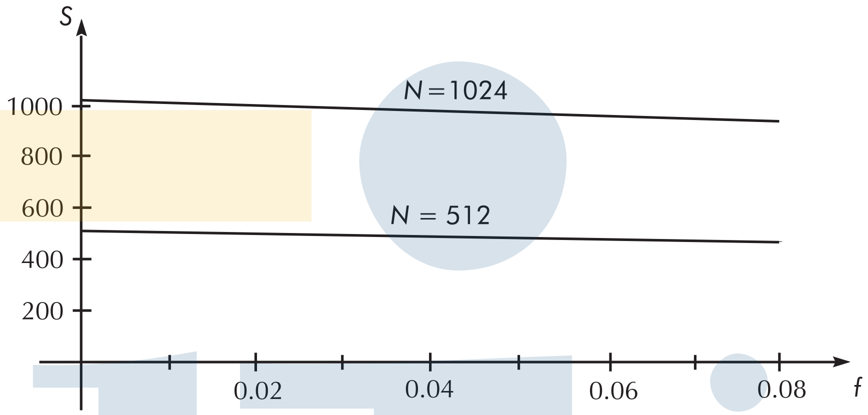

Fig. 1.3. Representación de la ley de Gustafson para 512 y 1024 procesadores.

to del volumen de cálculo es todavía mayor. Evidentemente, esto afecta, en general, a la parte paralelizable del problema y no a su parte secuencial, o al menos no en la misma medida. Si suponemos que el número de procesadores crece indefinidamente de la misma forma que las dimensiones del problema tendremos que

$$\lim_{N\to\infty} f = \lim_{N\to\infty} \frac{s}{s+Np} = 0 \qquad [1.12]$$

siendo s y p, respectivamente, las partes secuencial y paralela del problema antes de ser aumentado en relación al número del procesadores (por ejemplo, incrementando el número de puntos de la malla).

Con estas premisas, podremos calcular ahora la ganancia de velocidad para esta nueva situación (la parte paralela del problema ha crecido en la misma proporción que el número de procesadores). Para calcular la ganancia de velocidad supondremos que el tiempo que se tardaría en ejecutar el programa (ya incrementado) en un monoprocesador es:

$$t(1) = s + Np$$

y en un sistema paralelo sería:

$$t(N) = s + p$$

Por tanto, la ganancia en velocidad vendrá dada por:

$$S = \frac{t(1)}{t(N)} = \frac{s+Np}{s+p} \qquad [1.13]$$

que, teniendo en cuenta la definición de la fracción no paralelizable dada por 1.8, se podrá escribir como

$$S = f + N(1-f) = N - (N-1)f \qquad [1.14]$$

En la figura 1.3 puede verse la representación de la ecuación 1.14 para sistemas con diferente número de procesadores.

Podría pensarse que hay una aparente contradicción entre las leyes de Amdahl y Gustafson. Esto no es así debido a que las premisas de ambas leyes son distintas: la ley de Amdahl se refiere a procesos con un volumen de cálculo fijo mientras que la ley de Gustafson se refiere a problemas cuyo volumen de cálculo puede aumentar según el número de procesadores (esto se suele denominar **escalado** del problema). Afortunadamente, esta última situación es más cercana a la realidad.

1.3.4. Dependencias

Para que varios procesos puedan ejecutarse en paralelo, es necesario que dichos procesos sean independientes entre sí[1]. Esto, sin embargo, no siempre es así, debido a que existen **dependencias** entre los procesos. Estas dependencias pueden adquirir diferentes formas como se verá a continuación:

Dependencias de datos: se refieren a situaciones en que el orden en que se obtienen los datos marca el orden de ejecución de los procesos. Los tipos de dependencias de datos son los siguientes:

- **Dependencias de flujo**: un proceso P_2 es dependiente por flujo de otro proceso P_1, si P_2 sigue en el orden de programa a P_1, y una salida del proceso P_1 se utiliza como entrada en el proceso P_2.
- **Antidependencias**: un proceso P_2 es antidependiente de otro proceso P_1, si P_2 sigue en el programa a P_1 y una salida de P_2 se utiliza como entrada en P_1.
- **Dependencias de salida**: dos procesos son dependientes por la salida si escriben en la misma variable.
- **Dependencias de entrada/salida**: este tipo de dependencias pueden producirse cuando dos procesos acceden a un mismo fichero o dispositivo para leer o escribir datos
- **Dependencias desconocidas**: existen situaciones en que hay peligro de dependencias de datos, pero no es posible conocerlas antes de la ejecución. Un ejemplo de este tipo de casos son las instrucciones que afectan a variables cuyos subíndices están, a su vez, subindexados.

Dependencias de control: estas dependencias se producen cuando el orden de ejecución de los procesos no se puede conocer hasta el momento de la ejecución. Esto afecta a la ejecución de instrucciones o procedimientos afectados por instrucciones condicionales.

Dependencias por recursos: este tipo de dependencias se deben a conflictos en el uso de recursos compartidos. Estos recursos pueden ser recursos de almacenamiento, tales como áreas de memoria compartida, recursos de ejecución, como unidades aritméticas, etc.

Condiciones de Berstein

Berstein (1966) descubrió una serie de condiciones para que dos procesos pudieran ejecutarse en paralelo. El estudio de estas condiciones requiere algunas definiciones previas:

[1]A partir de este punto, la palabra *proceso* debe interpretarse en su sentido más general, es decir, cualquier fragmento de un programa en ejecución, esto puede referirse a cualquiera de los niveles descritos en la sección 1.2

Definiremos el **conjunto de entrada**, **conjunto de lectura** o **dominio** de un proceso P_i, como el *conjunto de todas las variables de entrada necesarias para la ejecución de ese proceso*. A este conjunto lo denotaremos por I_i.

De forma similar, se llama **conjunto de salida**, **conjunto de escritura** o **rango** de un proceso P_i, al *conjunto formado por todas las variables modificadas después de la ejecución de* P_i. A este conjunto lo denominaremos O_i.

Ambos conjuntos, el de entrada y el de salida, están formados por operandos o resultados que pueden residir en registros del procesador, en memoria, o incluso en ficheros.

Sean dos procesos P_1 y P_2, cuyos conjuntos de entrada son I_1 e I_2 y cuyos conjuntos de salida son O_1 y O_2, respectivamente. Estos dos procesos se podrán ejecutar en paralelo, y se denotará mediante $P_1 \| P_2$, si se verifican las siguientes condiciones:

$$\begin{aligned} O_1 \cap I_2 &= \emptyset \\ I_1 \cap O_2 &= \emptyset \\ O_1 \cap O_2 &= \emptyset \end{aligned} \qquad [1.15]$$

Estas ecuaciones son conocidas como **condiciones de Berstein** y expresan la independencia de flujo, anti-independencia e independencia de salida de los procesos P_1 y P_2.

Contrariamente a lo que pudiera pensarse, **la relación de paralelismo definida por las condiciones de Berstein no es una relación de equivalencia** puesto que, aunque verifica la propiedad simétrica, no cumple la propiedad reflexiva ni la transitiva.

Los resultados de dos procesos que pueden ejecutarse en paralelo son los mismos si se ejecutan secuencialmente, en cualquier orden, o si se ejecutan de forma paralela.

En general, un conjunto de procesos, $P_1, P_2, ...P_n$, pueden ejecutarse en paralelo, y se representará $P_1 \| P_2 \| ... \| P_n$, si, y sólo si, las condiciones de Berstein se cumplen con todas las parejas de procesos. es decir,

$$P_1 \| P_2 \| ... \| P_n \iff P_i \| P_j \;\; \forall i \neq j$$

Veamos ahora un ejemplo de aplicación de las condiciones de Berstein:

Ejemplo 1.2: *Consideremos el caso más sencillo en que los procesos son instrucciones simples. Sean los siguientes procesos:*

$$\begin{array}{ll} P_1 & w = a * b \\ P_2 & x = a + w \\ P_3 & y = b - w \\ P_4 & z = a - b \end{array}$$

Determinaremos ahora los conjuntos de entrada y salida de cada uno de los procesos:

$$\begin{array}{ll} I_1 = \{a, b\} & O_1 = \{w\} \\ I_2 = \{a, w\} & O_2 = \{x\} \\ I_3 = \{b, w\} & O_3 = \{y\} \\ I_4 = \{a, b\} & O_4 = \{z\} \end{array}$$

y ahora comprobaremos las condiciones de Berstein:

$$\begin{array}{llll}
I_1 \cap O_2 = \emptyset & I_2 \cap O_1 = \{w\} & O_1 \cap O_2 = \emptyset & \Longrightarrow P_1 \not\| P_2 \\
I_1 \cap O_3 = \emptyset & I_3 \cap O_1 = \{w\} & O_1 \cap O_3 = \emptyset & \Longrightarrow P_1 \not\| P_3 \\
I_1 \cap O_4 = \emptyset & I_4 \cap O_1 = \emptyset & O_1 \cap O_4 = \emptyset & \Longrightarrow P_1 \| P_4 \\
I_2 \cap O_3 = \emptyset & I_3 \cap O_2 = \emptyset & O_2 \cap O_3 = \emptyset & \Longrightarrow P_2 \| P_3 \\
I_2 \cap O_4 = \emptyset & I_4 \cap O_2 = \emptyset & O_2 \cap O_4 = \emptyset & \Longrightarrow P_2 \| P_4 \\
I_3 \cap O_4 = \emptyset & I_4 \cap O_3 = \emptyset & O_3 \cap O_4 = \emptyset & \Longrightarrow P_3 \| P_4
\end{array}$$

Por tanto, $P_2 \| P_3 \| P_4$, $P_1 \| P_4$, $P_1 \not\| P_2$ *y* $P_1 \not\| P_3$.

○

Grafos de dependencia de datos

Las condiciones de Berstein son un poco engorrosas de aplicar, por lo que veremos ahora una forma más práctica de detectar el paralelismo de los procesos.

Un **grafo de dependencia de datos** es un *grafo dirigido cuyos nodos representan a procesos y cuyos arcos describen las relaciones de dependencia de datos entre ellos*.

Los grafos de dependencia de datos expresan las restricciones del secuenciamiento de las instrucciones por dependencias de datos y ayudan mucho a la paralelización de los programas. Ilustraremos estas ideas con un pequeño ejemplo:

Ejemplo 1.3: *Supongamos que debemos evaluar la expresión*

$$x = \frac{a * b + c}{a * b - d}$$

Un programa para hacerlo constaría de las siguiente secuencia de procesos:

$$\begin{array}{ll}
P_1 & u = a * b \\
P_2 & v = u + c \\
P_3 & w = u - d \\
P_4 & x = v / w
\end{array}$$

En un computador convencional, estos procesos deben ejecutarse de uno en uno, aunque no necesariamente en el orden indicado (los procesos P_2 *y* P_3 *podrían intercambiarse). Sin embargo, en un computador paralelo, los procesos* P_2 *y* P_3 *podrían ejecutarse concurrentemente siempre que antes se haya terminado de ejecutar el proceso* P_1*, debido a la dependencia de datos existente. Por otra parte, el proceso* P_4 *no puede ejecutarse si antes no se han terminado de ejecutar los procesos* P_2 *y* P_3*. En la figura 1.4 se muestra el grafo de dependencia de datos para el cálculo de la expresión anterior. Este grafo muestra de una manera clara las dependencias de datos existentes entre las operaciones.*

○

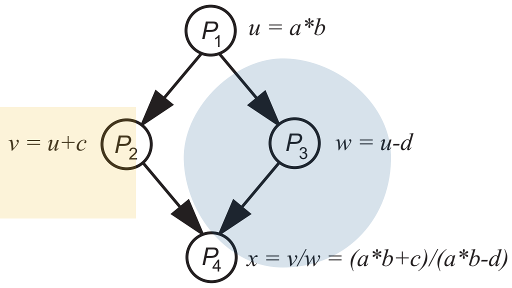

Fig. 1.4. Grafo de dependencia de datos para evaluar la expresión $x = (a * b + c)/(a * b - d)$.

1.4. Tipos de paralelismo

El paralelismo puede ser más o menos transparente al software que se ejecuta en el procesador. Si el paralelismo es totalmente transparente al software hablaremos de paralelismo implícito, en caso contrario, estaremos ante un paralelismo explícito. En otras palabras, el paralelismo implícito queda dentro del procesador, no trasciende al exterior y, por tanto, no afecta a la programación. El único efecto exterior que puede apreciarse en un procesador con paralelismo implícito es un incremento de la velocidad de procesamiento.

1.4.1. Paralelismo implícito

Las formas más habituales de llevar a la práctica el paralelismo implícito son la **segmentación** y la **división funcional**:

- Las **arquitecturas segmentadas** (también denominadas ***pipe-lines***) favorecen el encadenamiento del proceso de ejecución de las instrucciones, siendo un caso de paralelismo a nivel intrainstrucción. La segmentación divide a la función a realizar en una serie de subfunciones que se pueden ejecutar de forma independiente. Se diseñan unidades funcionales separadas para la ejecución de cada una de las subfunciones; de esta forma, se configura una cadena por la que van pasando la función a procesar. Las diferentes subfunciones pueden simultanearse en el tiempo, aunque sobre diferentes datos, de la misma forma que se simultanean los diferentes pasos de una fabricación en cadena.

  Los procesadores segmentados serán estudiados con detalle en el capítulo 2.

- La **división funcional**, que consiste en utilizar múltiples unidades funcionales, normalmente de tipo aritmético-lógico. De esta forma, sería posible ejecutar varias instrucciones al mismo tiempo (paralelismo a nivel de instrucción). Es bastante frecuente en computadores actuales, que exista una, o varias, ALU's para aritmética entera y otras separadas para aritmética de punto flotante; de esta forma, es posible poder ejecutar simultáneamente instrucciones enteras y otras de punto flotante. También puede ampliarse este concepto multiplicando el procesador, lo que da lugar a los procesadores superescalares que se estudiarán al final del capítulo 2.

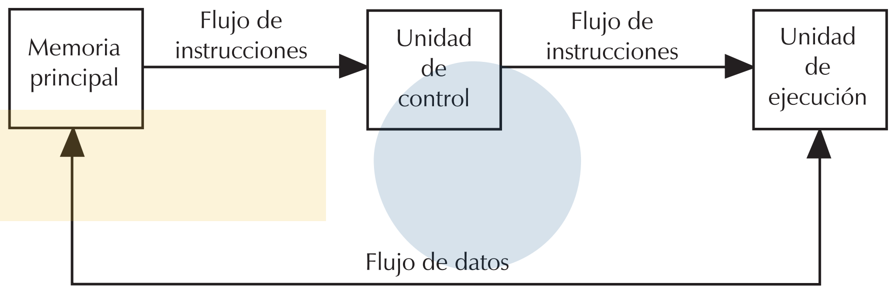

Fig. 1.5. Organización SISD.

1.4.2. Paralelismo explícito: clasificación de Flynn

Los sistemas con paralelismo explícito disponen de múltiples procesadores, o elementos de proceso, lo que hace necesario que sea el software del sistema quien decida qué instrucción debe ejecutarse en cada uno. Atendiendo a la multiplicidad de sucesos simultáneos que ocurren en los componentes de los sistemas de computadores, éstos pueden tener diferentes organizaciones según la clasificación de Flynn (1966):

SISD: **Flujo único de instrucciones y flujo único de datos (*single instruction flow, single data flow*)**, estos sistemas son los monoprocesadores convencionales con arquitectura de Von Neumann. La única posibilidad de efectuar paralelismo en este tipo de máquinas sería el paralelismo implícito. Un esquema general simplificado de esta organización, puede verse en la figura 1.5.

SIMD: **Flujo único de instrucciones y flujo múltiple de datos (*single instruction flow, multiple data flow*)**, en los sistemas con esta organización, varios procesadores ejecutan simultáneamente la misma instrucción, transmitida a través del flujo de instrucciones; sin embargo, cada procesador ejecuta esa instrucción sobre datos diferentes, que circulan por los diversos flujos de datos $FD_1, ..., FD_N$ (Ver figura 1.6). Este tipo de sistemas se conocen como **procesadores matriciales** (***Array Processors***) y son útiles en aplicaciones donde es necesario repetir la misma operación sobre datos diferentes. Un ejemplo de este tipo de aplicaciones es el cálculo vectorial.

MISD: **Flujo múltiple de instrucciones y flujo único de datos (*multiple instruction flow, single data flow*)**, esta organización, mostrada en la figura 1.7, se caracteriza por tener procesadores con unidades de control independientes que ejecutan diferentes flujos de instrucciones ($FI_1, ..., FI_N$) sobre el mismo flujo de datos. Algunos autores como Germain-Renaud y Sansonnet (1991) consideran a los procesadores segmentados como sistemas de este tipo; en general, esto no se considera así porque los procesadores segmentados están controlados por una sola unidad de control. Fuera de ello, este tipo de organización no se utiliza en la práctica.

MIMD: **Flujo múltiple de instrucciones y flujo múltiple de datos (*multiple instruction flow, multiple data flow*)** en este tipo de sistema existen N procesadores que, de forma simultánea, ejecutan instrucciones procedentes de los flujos $FI_1, ..., FI_N$ actuando, cada

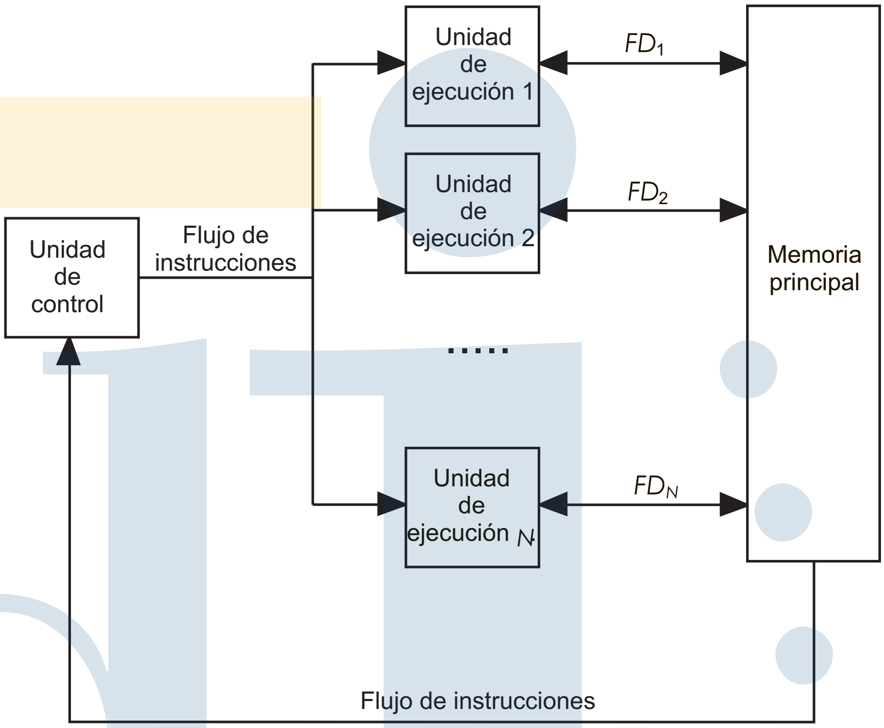

Fig. 1.6. Organización SIMD.

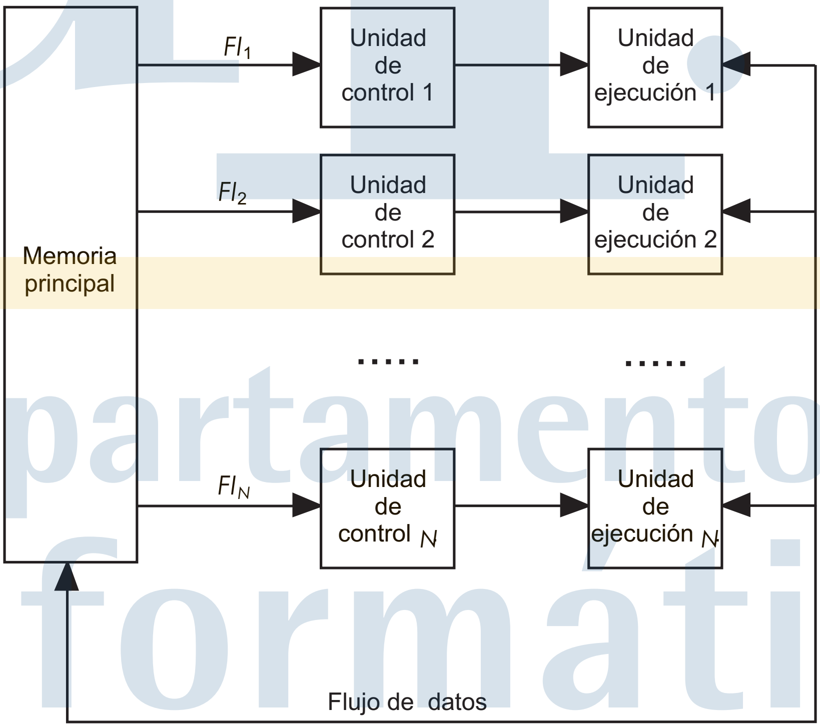

Fig. 1.7. Organización MISD.

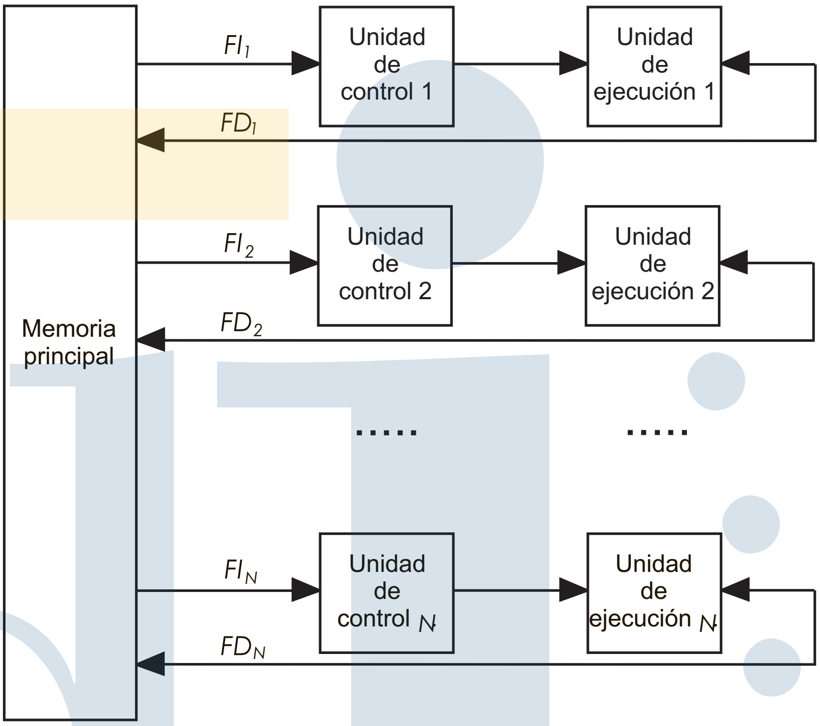

Fig. 1.8. Organización MIMD.

procesador, sobre cada uno de los diferentes flujos de datos $FD_1, ..., FD_N$. Un esquema de esta clase de sistemas se muestra en la figura 1.8. El paralelismo proporcionado por este tipo de sistemas admite explotar el paralelismo a los siguientes niveles:

- Ejecutar concurrentemente varias instrucciones de un algoritmo paralelizable.
- Procesar simultáneamente varios procedimientos independientes de un mismo proceso.
- Procesar concurrentemente procesos independientes.

La última posibilidad es la más utilizada en la actualidad (porque es la más fácil de implementar), pero cada vez se trata más de explotar las otras.

Un caso especial de sistemas MIMD lo constituyen los sistemas llamados **SPMD, único programa con múltiple flujo de datos (*single program, multiple data flow*)**. Estos sistemas son MIMD, pero cada procesador dispone de una copia diferente del **mismo** programa; esta copia se ejecuta sobre datos distintos en cada uno de los procesadores.

En la clasificación de Flynn, se supone que los procesadores no tienen memoria propia sino que acceden a una memoria central que contiene todos los datos y programas. Sin embargo, actualmente los progresos de la tecnología han permitido que cada procesador pueda poseer un memoria de gran capacidad. Por ello, los sistemas actuales no tienen la estructura expuesta en la clasificación de Flynn, aunque la idea subyacente siga siendo válida. Esto hace que exista otra clasificación que atiende a la posibilidad de compartir, o distribuir, la memoria entre los diferentes elementos de proceso. Según este criterio, los sistemas pueden ser:

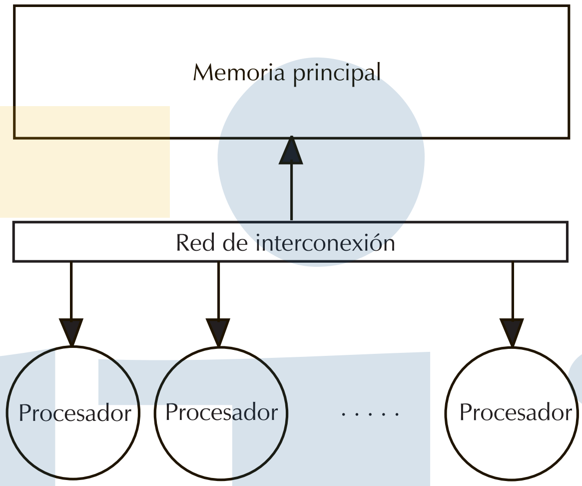

Fig. 1.9. Sistemas fuertemente acoplados.

Sistemas fuertemente acoplados: estos sistemas disponen de una memoria central a la que acceden todos los procesadores tal y como se muestra en la figura 1.9. Este tipo de sistemas también se denominan **sistemas con memoria compartida** o **multiprocesadores**.

Los inconvenientes de los sistemas fuertemente acoplados son:

- La memoria centralizada puede suponer un cuello de botella por confluir en ella todos los accesos a memoria solicitados por los procesadores.
- Al existir una memoria común, es necesaria una programación muy cuidadosa. Esto es debido a que la compartición de la memoria puede provocar interacciones no deseadas entre los programas que se ejecutan en los diferentes elementos de proceso.

Por el contrario, las ventajas de estos sistemas son las siguientes:

- La optimización de los recursos, ya que, al ser compartidos por los procesadores, pueden aprovecharse mejor.
- La programación, aunque es más delicada, también es más simple.

Sistemas débilmente acoplados: en esta clase de sistemas, cada uno de los elementos de proceso dispone de una memoria local en que almacena sus datos (figura 1.10). Estos sistemas también reciben el nombre de **sistemas distribuidos** o **sistemas con memoria distribuida**. En el fondo, los sistemas débilmente acoplados no son otra cosa que **redes de computadores** por lo que también se llaman **multicomputadores**.

Las ventajas de los sistemas débilmente acoplados son las siguientes:

- Cada elemento de proceso es completamente autónomo.
- No hay conflictos en la memoria por ser ésta distribuida.
- No pueden existir interacciones no deseadas entre los procesadores. Esto es debido a que la comunicación entre los elementos de proceso se efectúa exclusivamente por

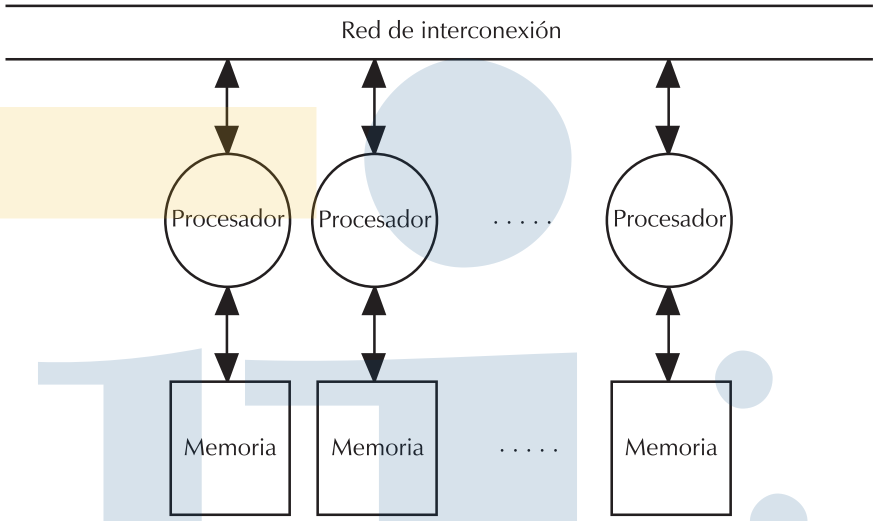

Fig. 1.10. Sistemas débilmente acoplados.

la técnica denominada **paso de mensajes** que consiste en comunicaciones puntuales entre los procesadores.

- Estos sistemas son muy fáciles de implementar porque las redes de computadores hoy en día están al alcance de todos.

El principal inconveniente de estos sistemas radica en la complejidad de la programación. Ello se debe a que todas las comunicaciones deben ser programadas explícitamente mediante paso de mensajes. Por otra parte, las cabeceras de los mensajes pueden cargar la red de interconexión existente entre los procesadores.

En la práctica, también puede haber casos con un grado de acoplamiento intermedio, en que cada procesador tenga una memoria local y, sin embargo, también tenga acceso a la memoria compartida. En otros casos, puede haber sistemas distribuidos en que haya un procesador principal que disponga de algunos recursos adicionales, por ejemplo, los periféricos.

En cuanto a la forma de gestionar el espacio de memoria, lo más frecuente es que los sistemas de memoria compartida tengan un espacio único de direcciones para todos los procesadores y, por el contrario, los sistemas con acoplamiento débil tengan un espacio de direcciones separado para cada procesador. Sin embargo muchos sistemas débilmente acoplados actuales disponen de modelos de memoria con un único espacio de direcciones para todos los procesadores. Estos modelos se denominan **DSM (*Distributed Shared Memory*, memoria compartida distribuida)** y **DVSM (*Distributed Virtual Shared Memory*, memoria distribuida virtualmente compartida)**; con ellos todos los procesadores tienen acceso, desde el mismo espacio de direcciones, a las memorias de todos los demás procesadores. Esto sólo ocurre desde el punto de vista de alto nivel. A más bajo nivel los accesos a las variables que están en una memoria que corresponde a otro procesador, se transforman en paso de mensajes. Esto tiene el fin de no tener los inconvenientes de los sistemas débilmente acoplados, es decir, la complejidad de la programación.

1.5. Problemática planteada

En este párrafo comentaremos los problemas planteados por los sistemas paralelos.

Uno de los principales problemas que se plantean es la forma de repartir el trabajo entre todos los elementos de proceso. Esto nos lleva a proponer las siguientes preguntas:

- ¿Cómo se pueden descomponer las tareas, que tradicionalmente se ejecutaban en un sistema monoprocesador, para que puedan ejecutarse en un sistema paralelo?

- ¿Podrá el software del sistema efectuar la descomposición anterior de forma automática o, por el contrario, deberá hacerla el programador explícitamente?

Otros problemas vendrán proporcionados por la forma de conectar los procesadores entre ellos y con la memoria. También puede pensarse si la red de interconexión ha de ser fija o puede ser dinámicamente reconfigurable.

En lo relativo al software, se deberán encontrar los algoritmos y lenguajes más adecuados para trabajar con las máquinas paralelas. Por otra parte, también se deberán diseñar los sistemas operativos apropiados para el proceso paralelo.

Por último, en algunos sistemas críticos, es necesario encontrar los medios para que el sistema siga respondiendo correctamente aunque falle alguno de los elementos de proceso (**sistemas tolerantes de fallos**).

1.6. Características de los sistemas paralelos

Detallaremos a continuación las definiciones de algunas características de los sistemas paralelos:

Grado de paralelismo: Se define el grado de paralelismo de un programa en un cierto instante t, $P(t)$, como *el número de elementos de proceso necesarios para explotar el paralelismo del programa de forma máxima en ese instante*.

La función $P(t)$ que nos da el grado de paralelismo en cada momento durante el tiempo de ejecución de un programa se llama **perfil de paralelismo** de ese programa.

Puede ocurrir que el grado de paralelismo de un programa, en algunos momentos, supere al número real de procesadores existentes en el sistema. En este caso, algunas partes del programa deberán ejecutarse disminuyendo el paralelismo teórico del algoritmo utilizado.

Llamaremos **máximo grado de paralelismo**, $P_{máx}$, al máximo valor del grado de paralelismo durante todo el tiempo de ejecución del programa, es decir:

$$P_{máx} = \text{máx}(P(t))_{t\in[t_1,t_2]}$$

donde t_1 y t_2 son, respectivamente, los instantes de comienzo y final de la ejecución del programa.

El **grado medio de paralelismo**, $\overline{P}$, se define como el valor medio del perfil de paralelismo durante todo el tiempo de ejecución del programa, es decir:

$$\overline{P} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} P(t)dt$$

Como $P(t)$ es una función discreta, es más fácil calcular el grado medio de paralelismo mediante la expresión

$$\overline{P} = \frac{\sum_{i=1}^{Pmáx} i.t^i}{\sum_{i=1}^{Pmáx} t^i} = \frac{\sum_{i=1}^{Pmáx} i.t^i}{t_2 - t_1} \qquad [1.16]$$

donde t^i es el tiempo durante el que se están utilizando i procesadores.

El número de instrucciones ejecutadas entre todos los procesadores de un sistema para llevar a cabo la ejecución de una tarea, se llama **carga de trabajo de la tarea** (W). Si un sistema está formado por suficientes procesadores de M MIPS y el tiempo viene dado en segundos, la carga de trabajo del proceso, en ese sistema vendrá dada por:

$$W = M \int_{t_1}^{t_2} P(t)dt$$

que, en la práctica, puede calcularse mediante la expresión

$$W = M \sum_{i=1}^{P_{máx}} it^i$$

Tasa de utilización del sistema (μ): se llama así a la relación entre el grado medio de paralelismo **explotado**, es decir, *referido al número de procesadores disponibles*, y el máximo grado de paralelismo, es decir,

$$\mu = \frac{\overline{P}_e}{P_{máx}} = \frac{\sum_{i=1}^{N} i.t^i}{(t_2 - t_1)P_{máx}} \qquad [1.17]$$

Donde N representa el número de **procesadores disponibles**.

Evidentemente, la tasa de utilización del sistema no dependerá sólo del número de procesadores sino también del programa que se esté ejecutando. Este parámetro nos facilitará la medida del aprovechamiento de los procesadores.

Grano de paralelismo: Se denomina grano de paralelismo al *número de elementos de proceso que componen un sistema paralelo*.

Si el grano es bajo se dice que la granularidad es **gruesa** y, si es alto, se dice que la granularidad es **fina**. En esta definición, el concepto de elemento de proceso es general y puede referirse tanto a un procesador completo como a una parte del mismo capaz de realizar una función elemental. Cuando nos queramos referir específicamente a un procesador completo con todos los elementos (unidad de control, ALU y memoria) hablaremos de **procesador elemental** o, abreviadamente, **PE** (*Processor Element*).

Escalabilidad: Un sistema se denomina **escalable** *si puede aumentarse su tamaño sin hacer grandes cambios y sin perder mucho rendimiento*. Dicho de otra manera, es la posibilidad de ampliar un sistema sin que ello suponga una devaluación de sus prestaciones. En concreto, en el caso de los sistemas paralelos, la escalabilidad se refiere al incremento del número de elementos de proceso. El grado de escalabilidad de un sistema lo podríamos evaluar por los siguientes factores:

- La facilidad con que el sistema pueda crecer, en el sentido de que se le puedan añadir elementos de proceso. Esto también afecta al coste del incremento de procesadores. El sistema será más escalable si el coste del sistema crece linealmente (y no más) con el número de elementos de proceso.
- La linealidad de la relación entre la ganancia de velocidad $S(N)$ y el número de elementos de proceso N. El sistema será tanto más escalable cuando la citada relación no esté muy por debajo de la linealidad.

Fiabilidad: Se define la fiabilidad de un sistema, en un instante t durante un tiempo Δt, como la *probabilidad de que ese sistema no tenga ninguna avería durante el intervalo de tiempo* $[t, t + \Delta t]$. Una forma más práctica de medir la fiabilidad es mediante un parámetro denominado $MTBF$ (*mean time between failures*, tiempo medio transcurrido entre dos averías consecutivas).

En lo referido a la fiabilidad, los sistemas pueden ser **redundantes** o **no redundantes**; en los últimos, para un buen funcionamiento del sistema es necesario que funcione correctamente cada uno de los componentes. En los sistemas redundantes, la duplicidad de algunos componentes o la totalidad del sistema, asegura el buen funcionamiento del mismo ante una avería de alguno de sus componentes.

Disponibilidad $A(t)$: Se llama disponibilidad de un sistema, en un instante determinado t, a la *probabilidad de que el sistema funcione correctamente en el instante* t.

Una forma práctica de medir este parámetro es calculando el porcentaje de tiempo que el sistema puede funcionar correctamente. Así la disponibilidad se puede evaluar mediante la siguiente fórmula:

$$A(t) = \frac{MTBF}{MTBF + MTTR}$$

donde $MTTR$ (*mean time to repair*) es el tiempo medio de reparación de una avería.

1.7. Aplicaciones del proceso paralelo

El procesamiento paralelo es aplicable en todos aquellos campos donde se necesite un gran volumen de cálculo y, por tanto, los computadores escalares tradicionales resulten insuficientes. Entre estas aplicaciones podemos destacar las siguientes:

1. Problemas de **cálculo numérico** muy voluminoso, en especial los relativos a **modelado y simulación** de sistemas. En este tipo de problemas incluye el modelado y simulación de la atmósfera, que persiguen elaborar predicciones meteorológicas; el análisis de elementos finitos, empleado en Ingeniería Civil (diseño de puentes, edificios, etc); y las simulaciones

del comportamiento de fluidos para diseñar la forma de algunos elementos externos de vehículos, tanto aéreos como terrestres.

En muchos de estos problemas se necesita hacer una simulación sobre una gran zona del espacio, en un amplio intervalo de tiempo, y, además, con una malla muy fina. Esto explica por qué el volumen de cálculo necesario es tan importante.

2. Problemas en **tiempo real**: procesamiento de imágenes, visión artificial, reconocimiento de patrones, etc. En este tipo de problemas, el tiempo disponible no es muy grande, y, sin embargo, sí lo es el volumen de cálculo necesario.

3. Aplicaciones de **diseño asistido por computador**: Los programas de diseño necesitan una gran cantidad de cálculo gráfico. El cálculo gráfico conlleva muchas operaciones complejas; para que estas operaciones puedan efectuarse en un tiempo razonable, es necesario que el sistema donde se procesan sea lo suficientemente rápido. Esto puede conseguirse mediante el uso de computadores paralelos.

4. Aplicaciones de **inteligencia artificial**. Estas aplicaciones, además de gran volumen de cálculo lógico, precisan acceder a bases de datos con numerosos registros, hacer búsquedas en ellas, etc. Esto induce al uso de computadores paralelos.

Bibliografía y referencias

AMDAHL, G. 1967. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *In: Proceedings AFIPS Conference*, vol. 30.

BERSTEIN, A.J. 1966. Analysis of Programs for Parallel Processing. *IEEE Transactions on Computers*, Oct.

FLYNN, M.J. 1966 (Dec.). Very High-Speed Computer Systems. *In: Proceedings of the IEEE*, vol. 54.

GAJSKI, D.D., & PEIR, J.K. 1985. Essential Issues in Multiprocessor Systems. *IEEE Computer*, **18**(6).

GERMAIN-RENAUD, C., & SANSONNET, J.P. 1991. *Les ordinateurs massivement paralleles*. Armand Colin Editeur. Existe traducción al castellano: Ordenadores masivamente paralelos, Paraninfo, 1993.

GUSTAFSON, J.L. 1988. Reevaluating Amdahl's Law. *Communications of the ACM*, **31**(5).

HWANG, K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill.

HWANG, K., & BRIGGS, F.A. 1984. *Computer Architecture and Parallel Processing*. McGraw-Hill. Existe traducción al castellano: Arquitectura de computadoras y procesamiento paralelo, McGraw-Hill, 1988.

ORTEGA, J. ET AL. 2005. *Arquitectura de Computadores*. Thomson.

STALLINGS, W. 2005. *Computer Organization and Arquitecture: Designing for Performance*. 7 edn. Prentice-Hall. Existe traducción al castellano: Organización y arquitectura de computadores, 7ª edición, Prentice-Hall, 2006.

CUESTIONES Y PROBLEMAS

1.1 Un computador tiene un procesador de 2 GHz. y ejecuta 1.500 MIPS, supongamos que cada acceso a memoria sólo requiere un ciclo de reloj:

a) ¿Cuál será el CPI de esta máquina?

b) Supóngase ahora que a la máquina anterior, se le actualiza el procesador con una nueva versión que puede funcionar con un reloj de 4 GHz. Suponiendo que la memoria del computador no cambia, y, por tanto, los accesos a memoria necesitarán ahora dos ciclos de reloj, calcular el CPI del nuevo procesador suponiendo que el 30 % de las instrucciones precisan dos accesos a memoria y el resto sólo uno.

1.2 Para un computador dado, calcular la relación existente entre su velocidad medida en MIPS, su frecuencia de reloj y el CPI, de forma que conocidos dos de esos datos se pueda calcular el restante.

1.3 Un procesador de 1 GHz. ejecuta un programa de prueba con la mezcla de instrucciones que se muestra en la tabla 1.1. Determinar el tiempo de ejecución del citado programa, el CPI de la máquina y su velocidad en MIPS.

1.4 Considerar la ejecución de un programa con 300.000 instrucciones en un procesador de 1 GHz. El porcentaje de cada tipo de instrucción en el citado programa, junto con el número de ciclos necesarios para su ejecución, se muestra en la tabla 1.2. Con estos datos, calcular:

a) El CPI y la velocidad en MIPS de esta máquina cuando se ejecuta el programa anterior.

b) El tiempo de ejecución del programa.

1.5 Supongamos ahora que el programa del problema anterior se ejecuta en un sistema con memoria compartida y 4 procesadores, y que las instrucciones se distribuyen uniformemente entre todos los procesadores. Supongamos también que, debido a la distribución y sincronización de los programas, el sistema operativo añade 3.000 instrucciones a cada procesador. Por otra parte, en el sistema paralelo, se puede suponer que las instrucciones que acceden a memoria, por fallo en caché, precisan, en media, 4 ciclos de reloj más debidos a conflictos en la memoria compartida. En estas condiciones, calcular:

Tabla 1.1.

Tipo de instrucción	Número de apariciones	Número de ciclos necesarios
Aritméticas enteras	12.000	2
Aritméticas de punto flotante	8.000	4
Transferencia de datos entre registros	20.000	1
Transferencia de datos a/desde memoria	10.000	2
Control de flujo	15.000	1

a) La velocidad en MIPS del sistema completo cuando se ejecuta el citado programa:

b) El tiempo de ejecución del programa con la máquina paralela.

c) Calcular la ganancia de velocidad de este sistema comparándola con la del problema anterior.

d) Calcular la eficiencia de este sistema.

1.6 Sea f el cuello de botella secuencial de un programa. Supongamos que ese programa se ejecuta en un sistema paralelo con N procesadores y que cada uno de esos procesadores funciona a M MIPS. Calcular la velocidad en MIPS del sistema paralelo para ese programa en función de los parámetros anteriores.

1.7 Supongamos un sistema con 32 procesadores en que se ejecuta un proceso que sólo puede ser paralelizable en un 60 %:

a) ¿Cuál es la ganancia máxima de velocidad que puede conseguirse en esas condiciones, respecto a la ejecución del proceso en un solo procesador?

b) ¿Cuál será la eficiencia máxima posible en ese sistema con el citado proceso?

1.8 Sea un problema cuya parte secuencial, que consta de E/S e inicializaciones de variables escalares, ocupa el 10 % del tiempo. El problema consiste en la resolución de una ecuación diferencial para un problema en el espacio-tiempo con una malla de 100 cortes por dimensión.

a) ¿Cuál será la ganancia máxima de velocidad posible para ese problema en un sistema con N procesadores?

b) El problema anterior se amplía aumentando la malla a 1000 cortes por dimensión (incluyendo como tal al tiempo) ¿Cuál será entonces la ganancia máxima de velocidad en el mismo sistema? ¿Y si el sistema se ampliara hasta $10N$ procesadores?

1.9 Supóngase que se quiere calcular las soluciones de una ecuación de segundo grado, de la que se sabe que tiene soluciones reales:

a) Separar este proceso en subprocesos elementales y aplicar sobre ellos las condiciones de Berstein para saber si pueden ejecutarse en paralelo.

Tabla 1.2.

Tipo de instrucción	Porcentaje	Número de ciclos necesarios
Aritméticas enteras	20 %	2
Aritméticas de punto flotante	10 %	4
Transferencia de datos entre registros	30 %	1
Transferencia de datos a/desde memoria (con acierto en memoria caché)	18 %	2
Transferencia de datos a/desde memoria (con fallo en memoria caché)	7 %	6
Control de flujo	15 %	1

b) Dibujar el grafo de dependencia de datos de los subprocesos resultantes del apartado anterior.

c) Dibujar el perfil de paralelismo para este proceso, suponiendo que cada operación elemental tarda en ejecutarse un ciclo de reloj y que la raíz cuadrada tarda tres ciclos en efectuarse.

d) En esas mismas condiciones, calcular el tiempo mínimo de ejecución de ese proceso si se dispone de tres procesadores.

e) Repetir el apartado anterior suponiendo que sólo se dispone de dos procesadores.

f) Para cada uno de los dos apartados anteriores calcular la ganancia, la eficiencia y la tasa de utilización de los sistemas.

1.10 Supóngase que se quiere evaluar la siguiente expresión:

$$p = \frac{(x^2 + y^2 + z^2)(z^2 - w^2)}{(x^2 + y^2 + w^2)(y^2 - z^2)}$$

a) Separar este proceso en subprocesos elementales y aplicar sobre ellos las condiciones de Berstein para saber si pueden ejecutarse en paralelo.

b) Dibujar el grafo de dependencia de datos de los subprocesos resultantes del apartado anterior.

c) Dibujar el perfil de paralelismo para este proceso, suponiendo que cada operación elemental tarda en ejecutarse un ciclo de reloj.

d) En esas mismas condiciones, calcular el tiempo mínimo de ejecución de ese proceso si se dispone de cuatro procesadores. ¿Y si sólo se dispusiera de 3?

e) Calcular la ganancia de velocidad y la eficiencia del sistema del apartado anterior cuando se evalúa la citada expresión.

f) ¿Cómo mejoraría el tiempo de ejecución del proceso si se dispusiera de ocho procesadores?¿Y la eficiencia del sistema?¿Y si sólo se dispusiera de tres?

1.11 Supongamos que un programa tiene el perfil de paralelismo que se muestra en la figura 1.11.

a) Calcular el paralelismo medio del citado programa.

b) Calcular la carga de trabajo de ese proceso si la gráfica de la figura se ha obtenido con procesadores que ejecutan 2 instrucciones por ciclo.

c) ¿Cuál será la ganancia máxima de velocidad que podrá conseguirse si se pudiera explotar el paralelismo al máximo?

d) ¿Cuál será la ganancia máxima de velocidad que podrá conseguirse cuando se ejecute ese programa en una máquina con 4 procesadores de la misma velocidad?

e) En las condiciones de los apartados anteriores: ¿Cuál sería la ganancia máxima de velocidad que se obtendría, en ese sistema, con un programa que tuviera un grado de paralelismo constante de 8?

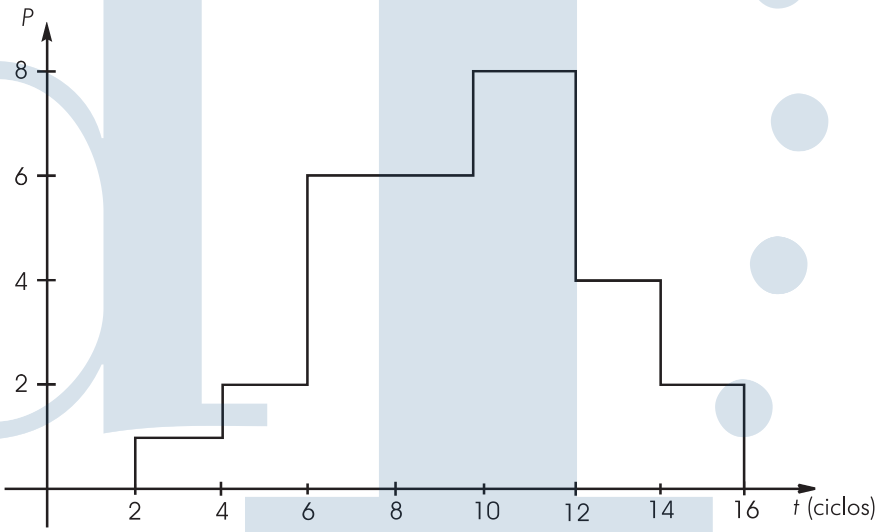

Fig. 1.11.