

INVESTIGACION

La sección de Difusión de Investigación en Ingeniería, como su nombre lo indica, pretende divulgar el trabajo de investigación y desarrollo que se haga en esta Facultad y otras Facultades de Ingeniería del país.

Esperamos que esta sección pueda servir para aumentar los mecanismos de comunicación de la comunidad científico-tecnológica en el país. Consecuentes con lo anterior invitamos a investigadores de otras universidades para que usen este espacio para divulgar resultados que sean de interés para un sector amplio de la ingeniería.

Rafael Gómez

Introducción

El Paralelismo o la Carrera a la Rapidez



Rafael Gómez

Ingeniero de Sistemas, Uniandes. DEA Informática, INPG, Grenoble, Francia. Profesor de Ingeniería de Sistemas. Área de especialización: Paralelismo.

Un refrán inglés pretende que el "el apetito viene comiendo"; este refrán se aplica muy bien en computación. En efecto, el desarrollo de la tecnología permite disponer de una mayor capacidad de cálculo y de almacenamiento de datos, lo cual conduce a desarrollar nuevas aplicaciones, que antes sencillamente habría sido utópico considerar. A su vez, las nuevas aplicaciones despiertan el deseo de desarrollar "más y mejor", hasta que llega un momento en que se copa la capacidad de la tecnología.

Por supuesto, estos desarrollos no se hacen por capricho; por el contrario, se trata de llenar las necesidades que anteriormente no

era posible satisfacer. Dicha imposibilidad puede venir de diferentes fuentes: falta de capacidad de almacenamiento, falta de algoritmos adecuados y falta de velocidad de cómputo.

"El paralelismo consiste en embarazar nueve mujeres al tiempo con la esperanza de que el niño nazca en un mes".

Anónimo.

En ocasiones, la velocidad de cálculo puede verse como una

comodidad más que una necesidad; por ejemplo, si se tiene un programa que se utiliza esporádicamente, una vez por mes o por semana, la diferencia entre que ejecute en un minuto o en diez puede no ser significativa. Pero en otros casos no es así: ¿cuál es la utilidad de un programa para predecir el clima que, a partir del clima de hoy, necesita un día para calcular el clima de mañana? Después de un día ya será mañana, y la predicción no pasará de tener un cierto interés académico.

Veamos el caso de la generación de imágenes por computador: si se gasta mucho tiempo en cada imagen, la producción de una película se demorará y aumentará los costos de desarrollo. Aun así es posible hacerla, pero, ¿qué ocurre si las imágenes son para un ingeniero, o un arquitecto, que está trabajando en un diseño de manera interactiva? Si la generación de una imagen toma 20 minutos, el trabajo será interactivo sólo nominalmente.

Una de las formas de aumentar la

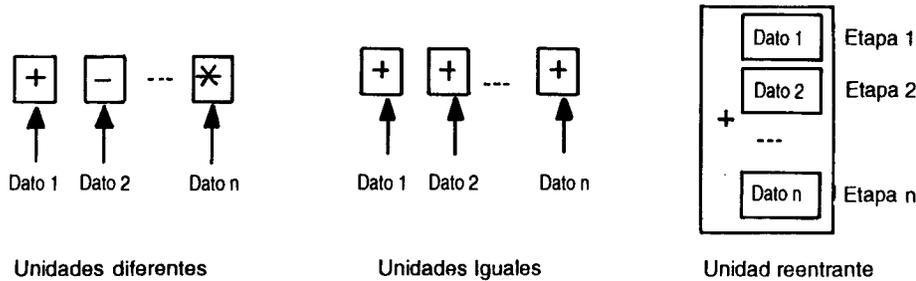


Figura 1.

velocidad de un programa es el paralelismo, que consiste en realizar varios cálculos al mismo tiempo con el fin de disminuir el tiempo real requerido.

El paralelismo no viene sin inconvenientes, que se pueden ejemplificar con el típico problema de regla de tres "si un obrero construye un cuarto en cien días, ¿en cuántos lo construyen 100 obreros?"; la respuesta es muy sencilla: "primero explique cómo meter 100 obreros entre un cuarto, y cómo hacer para sincronizarlos." De manera similar, si varias entidades están realizando cálculos, se pueden presentar problemas de dependencia de datos y de sincronización entre ellas. Estos problemas se pueden ilustrar con un par de ejemplos simples:

- ¿Cómo efectuar la suma $X + Y + Z$ en paralelo? ciertamente se puede calcular $X+Y$, pero para adicionar z es necesario contar de antemano con el resultado anterior.
- Se quiere calcular $e^X + e^Y$, sólo que X y Y son expresiones calculadas por otros procesadores. ¿Cómo sabe el encargado de calcular los exponentiales cuándo están listos los valores de X y Y ?

El problema de la velocidad de cómputo es muy viejo y ha sido, y es, atacado de diversas formas; el paralelismo es sólo una de ellas. A continuación se presentan algunas de manera esquemática.

¿Cómo Se Puede Acelerar Un Proceso?

Una forma de acelerar una acción

es hacer que los procesos físicos que intervienen transcurran más rápido. En el caso de los computadores, esto quiere decir que los componentes electrónicos deben reaccionar más rápidamente, lo cual se puede lograr de dos maneras: mejorando las técnicas actuales o desarrollando otros métodos que hacen lo mismo pero toman menos tiempo.

Se trabaja activamente en estos dos campos. De mejoras a la tecnología, se oye hablar con frecuencia: aumento del nivel de integración de los circuitos, semiconductores novedosos, otras formas de construir los circuitos integrados, etc.. En cuanto a métodos nuevos, hace ya varios años que se habla de computadores superconductores y, más recientemente, de computadores basados en luz.

No se puede negar que los avances en el primer frente son productivos; gracias a ellos se ha visto una mejora continuada en los últimos años. Pero dichas mejoras no son tan grandes como se quisiera. Además, la física cuántica ya lanzó su advertencia: se está llegando a los límites permitidos por la naturaleza.

Debido a lo anterior, se ha tratado de encontrar otras soluciones. La superconducción promete mucho, puesto que disminuye en buena medida el tiempo de conmutación. Sin embargo, la superconducción -o cualquier mejora técnica- presenta algunos inconvenientes pragmáticos: su desarrollo toma demasiado tiempo, por un lado, y los aumentos en velocidad son puntuales; es decir, cuando se empieza a utilizar la nueva tecnología, se logra un gran

aumento en la velocidad, pero mantener los aumentos es más difícil.

Otra forma de enfrentar el problema es el enfoque estructural; en lugar de concentrarse en acelerar los elementos físicos, se desarrollan organizaciones diferentes, se cambia la forma de funcionamiento. Veamos un caso similar en administración: una empresa puede aumentar su producción o reducir costos cambiando las máquinas utilizadas, lo cual corresponde a aumentar la velocidad del medio físico; pero también puede cambiar de métodos, tal como lo hizo Henry Ford cuando desarrolló la producción en línea.

Una forma evidente para volver más eficiente un proceso consiste en realizar simultáneamente el mayor número posible de acciones. Tal es el objetivo del paralelismo.

Cabe anotar que no todas las mejoras estructurales implican paralelismo; por ejemplo, los "cache" de memoria disminuyen el tiempo de respuesta de la memoria pero no usan técnicas paralelas.

El paralelismo presenta dos ventajas importantes. En primer lugar, es una solución que se mantiene válida independientemente de la tecnología utilizada; por ejemplo, si se desarrolla un computador superconductor, se podrá fabricar un computador superconductor paralelo. Además, es una solución graduable: si un programa requiere más velocidad, se puede intentar incrementar el número de acciones en paralelo. Por supuesto, esta última afirmación tiene límites; por ejemplo, si se quiere sumar dos vectores de

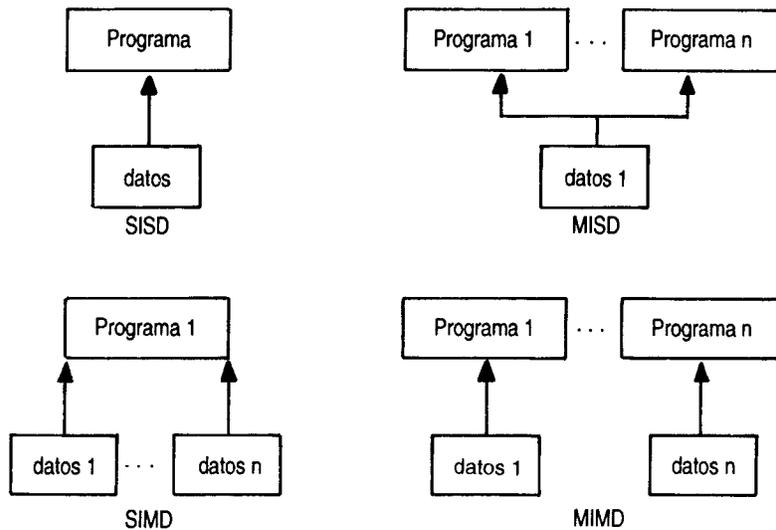


Figura 2

tamaño n , el máximo posible de paralelismo es realizar las n sumas al tiempo; no es posible encontrar el resultado en un tiempo menor.

Técnicas Generales Del Paralelismo

Grosso modo, se puede hablar de tres técnicas generales, que se ilustran en la fig. 1:

- *Múltiples unidades diferentes* en lugar de tener una sola unidad encargada de varias operaciones, se tienen diferentes unidades, cada una encargada de una operación. Por ejemplo, en lugar de tener un módulo encargado de las cuatro operaciones aritméticas, se provee un módulo para cada una de ellas; de esta manera se pueden calcular varias operaciones al mismo tiempo sobre diversos datos. Por supuesto, las operaciones no pueden ser la misma; v.g., no es posible hacer dos sumas al mismo tiempo.
- *Múltiples unidades iguales.* al replicar las unidades, es posible ejecutar, al mismo tiempo, la misma operación sobre varios datos.
- *Unidades reentrantes* ("pipeline"): la operación que realiza un módulo se puede

partir en etapas. Esto permite ejecutar la misma operación sobre varios datos, sólo que en diferentes etapas. Es la misma idea de las líneas de producción.

Por ejemplo, al multiplicar dos números de punto flotante, $(A \cdot 2^a) \cdot (B \cdot 2^b)$, hay que efectuar varias acciones: hay que multiplicar las mantisas, sumar los exponentes, redondear el resultado y ponerlo en una forma normal de representación. La unidad encargada de la multiplicación se divide en diversas etapas que realizan las acciones de manera secuencial, y se tiene un dato en cada etapa. Cuando una operación termina una etapa, pasa a la etapa siguiente. Esto permite tener múltiples operaciones ejecutando al mismo tiempo, sólo que en estados diferentes; algunas más adelantadas que otras.

Estas técnicas se aplican en tres niveles:

- Al interior de una instrucción: como cada instrucción de un computador implica una serie de acciones, se puede intentar paralelizarlas.
- Al interior de una secuencia de instrucciones: si una instrucción no depende de la anterior, se puede intentar ejecutarlas traslapadas, o incluso al mismo tiempo.

- Al interior de un programa: un programa puede estar compuesto de diferentes acciones que, aunque relacionadas, no necesariamente deben efectuarse de manera secuencial. Los segmentos de programa que se ejecutan simultáneamente se llaman procesos.

Como se dijo anteriormente, éstas son técnicas generales y se pueden utilizar de diversas maneras. Además, en una máquina concreta, se puede aplicar la misma técnica, o diferentes, en cada uno de los tres niveles. Por esta razón, al describir las técnicas, se habló de "operaciones"; dichas operaciones pueden ser funciones matemáticas (suma, resta, etc.) pero también pueden referirse a la ejecución misma de una instrucción; i.e., se puede tener una unidad reentrante para calcular sumas, como también se puede tener para ejecutar instrucciones de manera traslapada. Veamos unos cuantos ejemplos.

- Una sola instrucción se puede encargar de efectuar la misma operación sobre varios datos, lo cual se puede lograr usando una unidad reentrante o replicando la unidad encargada de la operación.
- Una sola instrucción se puede

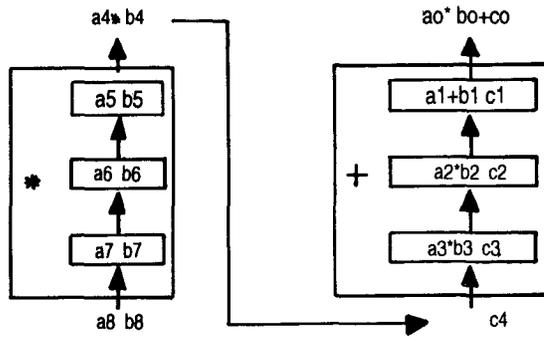


Figura 3

encargar de efectuar diversas operaciones. Esto requiere varias unidades independientes.

- Varias instrucciones pueden ejecutarse al mismo tiempo. Esto requiere que la unidad encargada de ejecutar las instrucciones esté replicada o sea reentrante. Lo mismo puede ser necesario para la unidad encargada de calcular la operación en sí.
- Varios procesos pueden ejecutarse al tiempo. Es necesario replicar el procesador.
- Un programa realiza la entrada/salida en paralelo. Hay que tener procesadores independientes especializados en entrada/salida.

Máquinas Paralelas

Se puede afirmar que la computación paralela no es una tecnología asentada. Todavía hay áreas muy oscuras donde no se sabe qué dirección tomar.

Una consecuencia es la gran cantidad de arquitecturas paralelas que se encuentran en diferentes etapas de desarrollo. Esto vuelve difícil clasificar las

arquitecturas paralelas; problema al que hace referencia la frase de Tanenbaum.

"Desafortunadamente, el Carlos Linneo de la computación paralela no ha surgido todavía".

Andrew S. Tanenbaum (TANE90)

En lo que sigue se presentan las arquitecturas más sobresalientes siguiendo el ordenamiento - más que clasificación- propuesto en (DECE89). Se han introducido algunas variaciones con respecto al esquema original.

En primer lugar, las arquitecturas se pueden dividir en arquitecturas tipo von Neumann y no-von Neumann. La arquitectura von-Neumann es la tradicional de los computadores: unidad de control, unidad aritmética-lógica, memoria y dispositivos de entrada/salida.

A- Arquitecturas tipo von Neumann:

Las arquitecturas tipo von Neumann, a su vez, se pueden dividir siguiendo el esquema

propuesto por Flynn. Dicho esquema divide los computadores siguiendo dos criterios: en el primero, se mira si el computador tiene una o varias secuencias de instrucciones independientes; el segundo, es igual pero aplicado a los datos. La clasificación, entonces, tiene en cuenta: "Single instruction vs. Multiple Instruction" y "Single Data vs. Multiple Data". Lo cual produce cuatro categorías: SISD, SIMD, MISD Y MIMD (figura 2), que se presentan a continuación, excluyendo MISD, de la cual no hay ejemplos reales.

1- SISD: una sola secuencia de datos y una sola se instrucciones. Corresponde a las máquinas tradicionales. Estas máquinas utilizan algunas técnicas paralelas pero en escala muy moderada.

En primer lugar, poseen "pipeline" de instrucciones; es decir, son capaces de procesar varias instrucciones al mismo tiempo, pero cada una está en una etapa diferente de ejecución. El "hardware" se encarga de evitar las colisiones entre instrucciones; un ejemplo, utilizando lenguajes de alto nivel, son las siguientes instrucciones:

A:= 2

B:=A*2

La segunda instrucción no puede continuar la ejecución mientras la

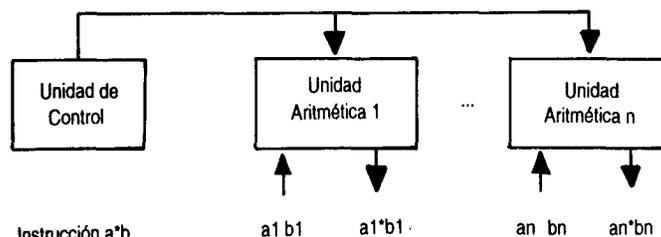


Figura 4

primera no haya terminado, ya que utiliza el valor de "A".

En segundo lugar, empiezan a surgir máquinas en las cuales una sola instrucción puede efectuar varias acciones al mismo tiempo (VLIW, "Very Long Instruction Word"). En este caso, es responsabilidad del compilador "armar" las instrucciones; por ejemplo, en las instrucciones:

A:= I*J

B:= X + Y

el compilador puede generar 3 instrucciones de máquina: la primera lee los valores de "I" y de "J" de la memoria; la segunda multiplica dichos valores y, al mismo tiempo, lee los valores de "X" e "Y"; la tercera suma los dos valores y almacena en memoria el resultado de la multiplicación.

Por último, algunas máquinas tienen procesadores especializados esclavos del procesador principal. Los procesadores especializados se encargan, típicamente, de las labores de entrada/salida.

2- SIMD: una sola secuencia de instrucciones y varias de datos. Se puede resumir diciendo que una instrucción opera sobre varios datos al tiempo, realizando la misma operación sobre todos. Normalmente se usa para procesar vectores. En esta categoría suelen estar las máquinas conocidas como "supercomputadores".

Se distinguen dos tipos de máquinas:

a- Procesadores "pipeline" o procesadores vectoriales. Las unidades aritméticas son reentrantes ("pipeline") y facilitan mucho el procesamiento de vectores; aquí reside esencialmente su poder computacional. Estas máquinas tienen instrucciones cuyos operandos son vectores; por ejemplo, la instrucción:

V:= C+ A*B

(* V(i)= C(i) + A (i)* B(i) *)

donde V,A,B y C son vectores se puede realizar con dos instrucciones de máquinas

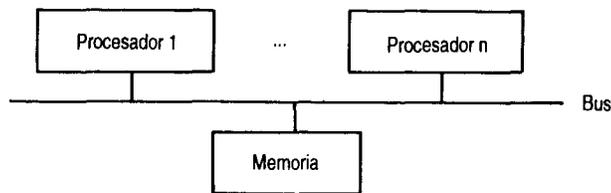


Figura 5

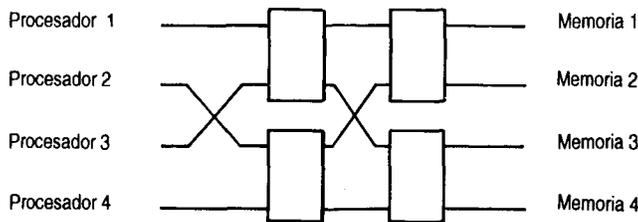


Figura 6

(figura 3). Es labor del compilador descubrir las posibles operaciones vectoriales; en algunos casos, los compiladores son capaces de detectar que un ciclo se puede convertir en una o más instrucciones de máquina sin necesidad de iterar.

b- Arreglos de procesadores: una sola unidad de control maneja varias unidades aritméticas. La unidad de control ejecuta las operaciones sobre escalares; las instrucciones vectoriales se distribuyen a todas las unidades aritméticas, de manera que cada una de ellas opera sobre un elemento de un vector (figura 4). La programación de estas máquinas es difícil y puede requerir algoritmos bastante truculentos. De nuevo, los compiladores se pueden encargar de paralelizar ciclos.

3- MIMD: varias secuencias de instrucciones y de datos. Este tipo

de máquina dispone de varios procesadores, cada uno de los cuales tiene su programa, pero colaboran en la solución de un mismo problema.

La programación de estas máquinas es ardua por cuanto es responsabilidad enteramente del programador; justamente, una de las áreas de investigación más importantes es la simplificación del desarrollo de programas.

Se encuentran varias clases de máquinas:

a- Bus común: es una extensión de las máquinas actuales. Todos los procesadores usan el mismo camino de comunicación con la memoria (bus), como se ilustra en la figura 5. Al utilizar el mismo camino, se crea una congestión en el acceso a la memoria, lo cual impide tener muchos procesadores; con más de 30 procesadores, el tiempo de acceso se degrada notablemente.

b- Red de conmutación: estas máquinas tratan de aligerar el "cuello de botella" en la memoria. La idea es tener varios caminos entre los procesadores y la memoria, para esto se construye una red cuyos nodos son elementos de conmutación capaces de enrutar los accesos a memoria; un ejemplo se ilustra en la figura 6.

Este esquema permite el uso simultáneo de la memoria por varios procesadores, pero impone algunas restricciones; por ejemplo, si dos pedidos están en el mismo módulo, uno de ellos debe esperar.

c- Conexión directa: otra forma, más radical, de eliminar la congestión en la memoria. Los procesadores no comparten la memoria sino que se comunican directamente; cada uno tiene su propia memoria a la cual los otros no tienen acceso, se deben comunicar por medio de mensajes entre procesadores. Por supuesto, esto implica que los procesadores deben tener líneas directas de conexión.

El funcionamiento es el siguiente: el compilador convierte el programa en un grafo; durante la ejecución, hay un conjunto de procesadores que recorren el grafo evaluando en paralelo lo que sea posible. La gran ventaja de estas máquinas es que el paralelismo es responsabilidad del lenguaje y del "hardware", no del programador. Todavía se encuentran en estado experimental.

Uso Del Paralelismo

Hoy día, las máquinas y la programación paralelas se suelen mirar como cosas un tanto exóticas; como si sólo fueran aplicables en ciertos casos... y puede que sea cierto. Sin embargo, tal no es la intención de los arquitectos de computadores; el objetivo último es desarrollar máquinas más rápidas de uso general, y no necesariamente máquinas más rápidas para una tarea particular.

Es conveniente mostrar un par de ejemplos. Cuando se desarrolló el concepto de "cache" de memoria, o cuando se idearon los "pipeline" de instrucciones, no se estaba pensando en la NASA ni en las

lenguaje de alto nivel puede ejecutar en una máquina con "cache" o sin él, sin que el programador se aperceba; mientras que si se desea utilizar una máquina paralela, es necesario programar teniendo en cuenta esto, y a veces no es sencillo.

Lo anterior hace que programar máquinas paralelas sea costoso, ya que se debe disponer de personal especializado. Por lo mismo, el mercado es limitado, por lo cual las máquinas se vuelven caras. En consecuencia, los únicos que usan el paralelismo son aquellos que tienen una necesidad absoluta de velocidad y que disponen del dinero para obtenerla.

El problema de la programación se ataca en varios frentes:

- Una posibilidad es olvidarse de la programación paralela y concentrarse en los sistemas operativos paralelos. Es decir, dejar que los usuarios sigan programando de la misma manera, pero permitir que los computadores tengan múltiples procesadores, de manera que el sistema operativo pueda tener varios programas independientes ejecutando al mismo tiempo. Esto aumenta la velocidad de la máquina como un todo, pero no acelera los programas individuales.

- Otra forma consiste en esconder el paralelismo por medio del compilador, es él quien se encarga de servir de puente entre la máquina y el programador. El programador sigue utilizando lenguajes tradicionales, tal vez aumentados con algunas construcciones, y el compilador se encarga de generar el código paralelo.

Esta estrategia funciona más o menos bien con las máquinas SIMD, y de hecho así se usan los supercomputadores. Sin embargo, por un lado, el éxito no es total en las máquinas SIMD, y, por otro lado, las máquinas MIMD no se han dejado "domesticar" de esta manera.

- Otra posibilidad es idear

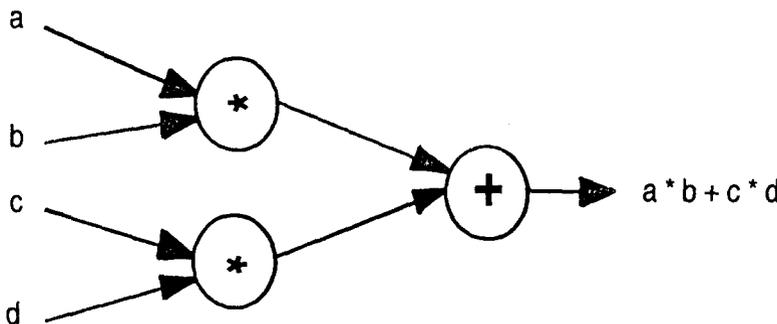


Figura 7

B- Arquitecturas no-von Neumann:

La arquitectura de "flujo de datos" ("Dataflow") es la más notable de esta clase. Básicamente consiste en crear un grafo que representa al programa y evaluarlo en paralelo. En la figura 7, se ve que las expresiones $a*b$ y $c*d$ pueden ser evaluadas en el paralelo.

compañías petroleras, sencillamente se quería que los computadores, cualquier computador, ejecutaran más rápido. Lo mismo se puede decir del paralelismo.

La diferencia entre el paralelismo y otras técnicas arquitecturales reside en que las técnicas paralelas son difíciles de esconder al programador. En efecto, un programa escrito en cualquier

nuevas arquitecturas que faciliten la extracción de paralelismo por parte del compilador. Esto es lo que se pretende con la arquitectura de flujo de datos. Pero, actualmente todavía quedan problemas lógicos y físicos para volver factibles estas máquinas.

- Por otro lado, existe la posibilidad de tomar el camino contrario de todas las alternativas anteriores; en efecto, todas intentan esconder el paralelismo. Se puede pensar en desarrollar técnicas de diseño y programación que incorporen la ejecución concurrente como una herramienta más. En este caso no se consideraría el paralelismo como una complicación sino como una ayuda. Esto implicaría que la idea no sería desarrollar el programa paralelo más eficiente sino el más cómodo para el programador. Es cierto que las máquinas no se usarían con todo su poder... pero al menos se podrían usar.
- Por último, queda la posibilidad más desagradable. Aceptar que la realidad no siempre es fácil, que si se quiere tener máquinas paralelas es necesario realizar

un trabajo complejo de programación. Por supuesto, las implicaciones son importantes: la programación y las arquitecturas paralelas deberían ser incorporadas como elementos regulares de los programas de ingeniería. Además, tendría implicaciones sociales, tales como actualizar a los ingenieros que ya ejercen. Todo esto hace poco atractivo, y poco viable, este camino.

En resumen, y especulando un poco, se podría decir lo siguiente: en un futuro inmediato se puede esperar un aumento en la cantidad de procesadores en las máquinas, utilizados a nivel del sistema operativo; es decir, diferentes procesadores para diferentes usuarios pero sin programación paralela. En la medida en que los intentos sean exitosos, muy probablemente se intentará mejorar la técnica pero manteniendo esencialmente el esquema actual; por ejemplo, se podría utilizar redes de conmutación.

Lo que ocurra después depende de muchos factores. Puede haber un cambio radical si se logra dominar alguna arquitectura; por ejemplo, si se resuelven los

problemas de las máquinas de flujo de datos, seguramente la tendencia será hacia ellas- debido al aumento en velocidad sin perjuicio para la programación-.

En caso de que no se logre obtener "la máquina maravillosa", y si los fabricantes y los usuarios están lo suficientemente desesperados, se podría empezar a usar la programación paralela sobre las máquinas paralelas a nivel del sistema operacional-bajo responsabilidad del programador-. Seguramente este cambio sería gradual, y, eventualmente, podría llevar a considerar la programación paralela como una habilidad necesaria, al menos para los ingenieros de sistemas. La situación será más simple si se logra desarrollar técnicas de desarrollo y programación que permitan incorporar la concurrencia como una herramienta.

Bibliografía

- (1) (DECE89) DECEGAMA, A. 1989. *The technology of parallel processing*, Prentice-Hall International. Editions.
- (2) (TANE90) TANENBAUM, A. 1989. *Structured computer organization*, Prentice-Hall International. Editions.

