EL PARALELISMO: UNA MANERA DE MEJORAR LA EFICIENCIA DE LOS ORDENADORES

José M. García Carrasco y Fco. José Quiles Flor
Departamento de Informática
Escuela Universitaria Politécnica de Albacete

Resumen

En la actualidad los ordenadores están invadiendo todos los campos del conocimiento. Cada vez mas, los procesos industriales de todo tipo están siendo controlados por los computadores, permitiendo que los hombres se puedan dedicar a tareas mas creativas. Para que esta sustitución del hombre por la máquina sea eficaz, la máquina debe tener un tiempo de respuesta muy rápido, poseyendo para ello una gran capacidad de procesamiento. En la actualidad, la solución a este problema descansa en el procesamiento paralelo. Esta técnica, hoy ya una realidad, está revolucionando el mundo de los grandes computadores, habiendose anunciado ya su llegada al mundo de los ordenadores personales. Desde luego, el mundo de la informática del año 2000 es a la fuerza un mundo paralelo.

1.- INTRODUCCION

Una batería de satélites en el espacio exterior recoge un rango de 10^{10} bits por segundo. Los datos representan información del tiempo meteorológico, la polución, la agricultura y recursos naturales. Para que esta información sea útil necesita ser procesada con una velocidad de al menos 10^{13} operaciones por segundo.

Por otra parte, un equipo de cirujanos desea ver en un dispositivo especial una imagen tridimensional del cuerpo del paciente en espera para cirugía. Necesitan rotar la imagen para obtener una sección transversal de un órgano, observándolo en detalle y ejecutar una cirugía simulada, todo ello sin tocar al paciente. Una velocidad de procesamiento mínima de 10^{15} operaciones por segundo es necesaria para realizar eficazmente los pasos descritos.

Los dos ejemplos anteriores son representativos de aplicaciones donde se necesitan ordenadores extremadamente rápidos para procesar grandes cantidades de datos o para permitir un gran número de cálculos muy rápidos (o al menos, dentro de un período de tiempo razonable). Otras clases de aplicaciones incluyen el testeo de aeroplanos, el desarrollo de nuevos medicamentos, exploración de petróleo, modelado de reactores de fusión, planificación económica, criptoanálisis, gestión de grandes bases de datos, astronomía, análisis biomédico, reconocimiento del habla en tiempo real, robótica y la solución de grandes sistemas de ecuaciones diferenciales parciales que surgen de simulaciones numéricas en disciplinas tan diversas como sismología, aerodinámica y física nuclear, atómica y del plasma. Todavía no existe ningún computador que pueda desarrollar la velocidad de procesamiento requerida por estas aplicaciones. Incluso los llamados *supercomputadores* alcanzan unos pocos billones de operaciones por segundo.

Durante los pasados 40 años se lograron incrementos asombrosos en la velocidad de computación. La mayoría de ellos, en su mayor parte, se deben al uso de componentes electrónicos más rápidos en la construcción de los ordenadores. Conforme se avanza desde las válvulas de vacío hasta los transistores y desde pequeña a mediana y larga escala de integración, aumenta la potencia de los ordenadores y por tanto el tamaño y rango de los problemas computacionales que podemos resolver.

Desafortunadamente es evidente que esta tendencia pronto tiene un final. El factor límite es una simple ley física que da la velocidad de la luz en el vacío. Esta velocidad es aproximadamente igual a $3*10^8$ m/s. Se asume que un periférico electrónico permite 10^{12} operaciones por segundo. Entonces tarda más una señal en viajar entre dos periféricos similares medio milímetro que lo que tardan en su proceso. En otras palabras, toda la ganancia en velocidad obtenida construyendo componentes electrónicos super-rápidos se pierde cuando un componente está esperando a recibir una entrada de otro. ¿Por qué entonces no se acercan más los componentes que se comunican?. De nuevo, la física dice que la reducción en distancia entre dos periféricos electrónicos alcanza un punto a partir del cual dichos periféricos empiezan a interaccionar, de esta forma se reduce no sólo su velocidad sino también su fiabilidad.

La única manera de resolver este problema es usar una nueva técnica avanzada denominada *paralelismo*. La idea es que si una tarea se descompone en una serie de operaciones, y estas operaciones se realizan simultáneamente, el tiempo que tarda en realizarse dicha tarea puede reducirse significativamente. Esta es una noción intuitiva y es a lo que se está acostumbrado en una sociedad organizada. Desde el correo hasta la vendimia y desde una oficina hasta el trabajo en una fábrica se ofrecen numerosos ejemplos de paralelismo a través de tareas compartidas.

Incluso en el campo de la computación la idea del paralelismo no es completamente nueva y ha adoptado muchas formas. Desde los primeros momentos del procesamiento de la información las personas se dan cuenta que es mucho más ventajoso tener muchos componentes de un computador que hagan diferentes cosas al mismo tiempo. Por ejemplo, mientras la CPU realiza cálculos, la entrada de datos puede ser leída de un cassette y la salida llevada a una impresora de líneas. En máquinas más avanzadas hay varios procesadores simples, cada uno de ellos realizando una tarea distinta. En nuestros días algunos de los más poderosos computadores contienen dos o más unidades de proceso que comparten los trabajos solicitados.

En cada uno de los ejemplos anteriormente mencionados el paralelismo se explota rentablemente. Estrictamente hablando ninguna de las máquinas mencionadas es verdaderamente un computador paralelo. En el moderno paradigma que queremos describir, sin embargo, la computación paralela se puede realizar en toda su potencia. La meta de estos nuevos diseños sería la siguiente: "Una gran colección de procesadores que puedan comunicar y cooperar para resolver grandes problemas rapidamente" [Alm89].

Al llevar a la práctica dicha idea, surgen una gran cantidad de interrogantes. Es dificil determinar el número y el tipo de los microprocesadores a usar, cómo debe ser la red de interconexión entre los diversos procesadores, cuál es el tamaño óptimo de cada subtarea a realizar en el procesador, etc.

A continuación pretendemos esbozar cuales han sido las soluciones adoptadas hasta ahora tanto desde un punto de vista de la arquitectura del ordenador como desde un punto de vista de la programación en paralelo.

2. CLASIFICACION DE LOS ORDENADORES.

Cualquier computador, ya sea secuencial o paralelo, opera por la ejecución de instrucciones sobre datos. Un flujo de instrucciones (el algoritmo) indica al computador qué es lo que tiene que hacer en cada paso. Un flujo de datos (la entrada del algoritmo) es modificado por estas instrucciones. Dependiendo de si hay uno o varios de esos flujos podemos distinguir entre cuatro clases de computadores [Fly72]:

1.- *Simple* flujo de *instrucción*, *simple* flujo de *datos* (acrónimo SISD, del inglés *simple instruction simple data*).

2.- Múltiple flujo de instrucción, simple flujo de datos (MISD).

3.- Simple flujo de instrucción, múltiple flujo de datos (SIMD).

4.- Múltiple flujo de instrucción, múltiple flujo de datos (MIMD).

A continuación vamos a describir brevemente cada una de estas arquitecturas paralelas. Para una mayor profundización, se puede ver [Hwa84] o [Dun90].

SISD

Un ordenador de esta clase contiene una única unidad de procesamiento, que recibe un flujo de instrucciones simple y que opera con un flujo de datos simple. En cada paso de la ejecución, la unidad de control proporciona una instrucción que opera con un dato obtenido de la unidad de memoria.

La mayoría de los computadores de hoy se ajustan a este modelo desarrollado por von Newman y sus colaboradores a finales de los años 40. Los algoritmos para computadoras de este tipo se denominan secuenciales.

MISD

En este caso, N procesadores cada uno con su propia unidad de control comparten una unidad de memoria común donde residen los datos. Hay N flujos de instrucciones y un sólo flujo de datos. En cada paso, se opera simultáneamente en cada procesador sobre un dato recibido de memoria, según el flujo de instrucciones recibidos desde su unidad de control. El paralelismo se logra dejando al procesador que haga diferentes cosas al mismo tiempo sobre los mismos datos. Este tipo de arquitectura no se ha desarrollado hasta ahora y hay algunos autores que piensan que es díficil llevarlo a la práctica.

SIMD

Esta arquitectura paralela consiste en N procesadores idénticos, cada uno con su propia memoria local donde guarda los datos. Todos los procesadores operan bajo el control de un flujo de instrucciones simple proporcionado por una unidad central de control.

Los procesadores operan **síncronamente**: en cada paso, todos los procesadores ejecutan la misma instrucción sobre diferentes datos. La instrucción puede ser simple (sumar o comparar dos números) o compleja (la mezcla de dos listas de números). De igual forma, los datos pueden ser simples (un número) o complejos (un conjunto de números). Los procesadores que están inactivos durante la ejecución de una instrucción o que completan la ejecución de la instrucción antes que los demás están desocupados hasta que se da la próxima instrucción.

MIMD.

Esta clase de computadores es la más general y la más potente en nuestro paradigma de clasificar las máquinas paralelas de acuerdo a si el flujo de datos o de instrucciones se duplica. Se va a disponer de N procesadores, N flujos de instrucciones, y N flujo de datos.

Cada procesador opera bajo el control de un flujo de instrucciones que le proporciona su propia unidad de control. Todos los procesadores ejecutan programas diferentes sobre diferentes datos mientras resuelven subproblemas de un único problema. Esto significa que los procesadores operan **asíncronamente**. De cara a la correcta resolución del problema, los diferentes procesos se tienen que comunicar y sincronizar. Esto se puede realizar mediante memoria compartida o por medio de una red de interconexión. Los computadores MIMD que comparten memoria común se les conoce como **multiprocesadores**, mientras que los que utilizan una red de interconexión se les conoce como **multicomputadores**.

Hasta aquí la exposición de la evolución de los ordenadores. Pero un ordenador para que sea efectivo se tiene que programar, y el arte de programarlo adecuadamente no está al alcance de cualquiera. En la siguiente sección se va a detallar cómo se puede abordar este problema.

3. DESARROLLO DE ALGORITMOS EN PARALELO.

Aunque el mundo en el que nos movemos es paralelo, y nuestra misma forma de vivir también es así (mientras escucho un disco, fumo un cigarro y al mismo tiempo leo un libro), el arte de la programación tradicionalmente ha sido secuencial.

El desarrollo de la programación en paralelo no se ha debido a un paso natural en el desarrollo del software, sino debido a una necesidad de aumento de potencia en los ordenadores.

El desarrollo de las nuevas arquitecturas que se han descrito en la sección anterior ha provocado la necesidad de disponer de unas herramientas para el desarrollo de programas en esas máquinas. Al igual que sucedió al inicio de los años 60 con los primeros ordenadores, nos encontramos con un vacío entre las arquitecturas desarrolladas y los lenguajes de programación para usar esas arquitecturas. De tal forma que, como viene recogido en [Kar87], podemos decir que *estamos programando las máquinas paralelas en el equivalente al lenguaje máquina*.

Si ya de por sí esto complica el diseño y desarrollo de algoritmos paralelos, hay otro factor aún más importante que lo dificulta, y es el comportamiento de un programa paralelo. A diferencia de un programa secuencial, un programa paralelo puede tener un comportamiento impredecible -no determinista-. Como dicen McGraw y Axelrod en [Gra88], *el hecho de que algún programa [paralelo] funcione correctamente una vez, o incluso cien veces, con algún conjunto de entradas determinado, no garantiza que no fallará mañana con las mismas entradas*.

El problema radica en que lo único que se busca es el aumento de la capacidad de procesamiento que ofrecen las máquinas con múltiples procesadores, pero no los efectos de la concurrencia, considerando el no determinismo de estas máquinas como un efecto lateral indeseable, más que como una propiedad que tiene que ser explotada.

Ello conduce a una ineficiencia por parte del programador, pues sigue planteándose la resolución de problemas con una mentalidad secuencial. Si distinguimos cuatro pasos en el desarrollo de un programa:

- Planteamiento del problema y elección de una estrategia para su resolución.
- Desarrollo de un algoritmo que resuelva el problema anterior.
- Implementación del algoritmo usando un lenguaje de programación.
- Compilación, ejecución y verificación del programa.

podemos advertir que la resolución paralela se plantea habitualmente en la tercera fase, continuando el planteamiento del problema y el desarrollo del algoritmo de una forma secuencial.

Debido a esto ha habido una proliferación de pseudo-lenguajes paralelos, consistentes en lenguajes secuenciales a los que se les ha añadido algunas extensiones (a modo de directivas de compilación o macros) para manejar el paralelismo. Ciertamente, esto permite seguir usando todo el software ya desarrollado con muy pocos cambios, además de no tener necesidad de aprender otro lenguaje de programación, pero empobrece la idea de la programación en paralelo.

Por todo ello es necesario un impulso al desarrollo de algoritmos paralelos desde una óptica paralela, así como la creación y uso intensivo de lenguajes realmente paralelos.

Junto a esto, y como también se comenta en [Kar87], es necesario el desarrollo de herramientas que permitan una cómoda programación en paralelo. Depuradores, compiladores, desarrollo de interfaces gráficas, etc., serán necesarios para el correcto uso y extensión entre la comunidad científica que se dedica a la programación paralela.

4. MEJORANDO LOS LENGUAJES.

Para poder programar adecuadamente un algoritmo para una máquina paralela, es necesario disponer de un lenguaje de programación que permita explotar al máximo todas las posibilidades de la nueva arquitectura paralela. Esta es la razón de que en los últimos años esté habiendo una auténtica proliferación de nuevos lenguajes. Para la creación de estos nuevos lenguajes, se puede seguir una de las cuatro vias siguiente [Gar89]:

1. Extender los lenguajes existentes, tales como Fortran o C, con nuevas operaciones que permitan a los usuarios expresar la concurrencia y la sincronización de tareas.
2. Mejorar los compiladores existentes para identificar las operaciones concurrentes, y añadir la sincronización necesaria al generar el código para mantener la corrección de los programas.
3. Crear un nuevo lenguaje que se utilizara sólo para expresar la concurrencia y la necesaria sincronización, mientras el resto de aplicaciones básicas de programación permanecerían iguales. Es decir, sería algo así como un pre-procesador para el manejo de la concurrencia.
4. Definir un nuevo lenguaje junto con su compilador que integre el concepto de concurrencia y sincronización junto con las estructuras de programación tradicionales.

Vamos a desarrollar cada una de estas vias, discutiendo las ventajas e inconvenientes que presenta.

Extensiones del lenguaje

Un camino muy utilizado hasta ahora ha sido el extender el lenguaje que se estaba usando con un mínimo conjunto de primitivas que permitan expresar el paralelismo. Las grandes ventajas que esto presenta son las siguientes:

a) A veces, las extensiones se pueden añadir mediante la incorporación de nuevas rutinas a las librerias de ese lenguaje, dejando el lenguaje original y su compilador sin necesidad de cambio alguno. Esto hace que sea fácil y barato el probar diferentes extensiones, pudiendo elegir las que parezcan mas adecuadas.

b) Un programa existente para una aplicación determinada puede ser paralelizado con mínimos cambios (medido en el número de lineas que han debido ser cambiadas).

Pero tambien aparecen dificultades. La mayoría debido a que el compilador no conoce cómo se está usando el paralelismo en el programa. Por lo tanto no puede ayudar en la optimización y depuración del programa. Por otra parte, el compilador podría realizar optimizaciones que en un entorno secuencial fueran correctas, pero que en un entorno paralelo perdieran su sentido, como por ejemplo mover código fuera de una región crítica.

Por último, otra dificultad que aparece es que es fácil escribir código que sea incorrecto y sin embargo que el compilador no detecte ningún error. Ello es debido a que sólo se asegura debilmente las construcciones de sincronización y el uso de datos compartidos.

Mejora de compiladores

Desde el punto de vista del escritor de programas, esta es la mejor solución. La idea sería que un programa funcionara igual (sin tener que hacer ninguna modificación en el texto fuente) tanto en un ordenador convencional como en un sistema paralelo. Tambien desde el punto de vista del amplio software ya existente es la mejor solución y la mas barata economicamente. El problema es que para ello se requiere un compilador muy sofisticado.

Logicamente, y debido a que la parte fundamental que entra en juego es la de generación de código, estos compiladores dependen mucho de la máquina objeto para la que se proyectan. Varios trabajos de investigación se han hecho ya en este area, tanto para máquinas vectoriales como para sistemas multiprocesadores. La mayoria de estos trabajos se han realizado en el análisis de bucles y lazos para encontrar la máxima cantidad de concurrencia.

Un lenguaje pequeño para la concurrencia

La idea es la siguiente: la mayoria del código que existe en la actualidad es correcto y sirve para resolver unos problemas dados. Este código para ser ejecutado en varios multiprocesadores no necesita cambiar en su gran mayoría, tan sólo añadirle algunas características de sincronización. De hecho, la mayoría de los errores en programación concurrente aparecen en la sincronización o partición de los datos. Por lo tanto, una posible solución es usar un pequeño lenguaje-compilador para expresar la estructura paralela del programa, dejando que el resto del código se exprese en un lenguaje convencional (Fortran o C).

Algunos trabajos iniciales se han hecho ya en este area. Pero esta solución tambien tiene sus inconvenientes. Uno de los mas importantes se refiere a la depuración de programas, ya que debido a la existencia de un pre-procesador es difícil detectar donde está el fallo cometido.

Nuevo diseño de un lenguaje/compilador

Se trata sin duda de las mas radical de las cuatro vías que se han comentado. Evidentemente, es la mas cara y la que exige mas labor de desarrollo. Como contrapartida, la mayoría de los problemas que se han expuesto hasta ahora se pueden solucionar mediante un diseño adecuado del sistema lenguaje/compilador.

Los avances realizados hasta la fecha se han orientado en el área de los lenguajes funcionales, como FP o SISAL. Las principales características de diseño de estos lenguajes son mejorar la claridad de la concurrencia, sin requerir que el programador maneje la sincronización, asi como permitir que no dependan de la arquitectura objeto sobre la que se van a implementar.

Otra de las grandes ventajas aparece desde el punto de vista de la depuración de programas. Por contra los mayores inconvenientes descansan en la construcción del compilador. Este requiere técnicas muy sofisticadas para realizar con eficacia las funciones que tiene encomendadas, como particionar, mapear y sincronizar el programa. En la actualidad se están teniendo problemas en el manejo dinámico de memoria, ya que las mas simples implementaciones requieren grandes cantidades de memoria.

5. CONCLUSIONES

En este artículo se ha presentado en qué estado se encuentra el mundo paralelo dentro de los ordenadores. Después de justificar el por qué de la necesidad de paralelismo, se han descrito las principales características arquitectónicas de los computadores paralelos. Junto a esto, se ha explicado cuales son los problemas a que se enfrenta el programador al utilizar este nuevo tipo de máquinas. Finalmente, se ha descrito cuales son las principales vias para el desarrollo de nuevos lenguajes de programación. Para el lector interesado, se recomienda [Bab88] donde se programa un algoritmo ejemplo con diferentes lengujes de programación, ofreciendo una panorámica interesante de los *pros* y *contras* de cada uno.

BIBLIOGRAFIA

[Alm89] Almasi, G.S. and Gottlieb, A. 1989. Highly Parallel Computing. Londres, The Benjaming/Cummings Publishing Company.

[Bab88] Babb II, R. G. 1988. Programming Parallel Processors. New York, Addison-Wesley Publishing.

[Dun90] Duncan, R. 1990. "A survey of parallel computer architectures". IEEE Computer, Vol. 23, (2), pp. 5-16.

[Fly72] Flynn, M.J. 1972. "Some computer organizations and their efectiveness". IEEE Trans. on Computers, Vol. C-21, (10), pp. 948-960.

[Gar89] García Carrasco, J.M. 1989. "Modelos de arquitecturas, lenguajes y compiladores para procesamiento en paralelo". Dept. de Ingeniería de Sistemas, Computadores y Automática. Univ. Politécnica de Valencia.

[Gra88] McGraw, J.R. and Axelrod, T.S. 1988. "Exploiting multiprocessors: issues and options." IEEE Software, Vol. 5, (5), pp. 7-25.

[Hwa84] Hwang, K. and Briggs, F. 1984. Computer Architecture and Parallel Processing. New York, McGraw-Hill.

[Kar87] Karp, A. 1987. "Programming for parallelism". IEEE Computer, Vol. 20, (5), pp. 43-57.