

NETWORKING CENTER®

Arquitecturas Paralelas

Autor: Miguel Angel Pérez (iceman)

Edición: Marzo, 2001

Indice de Contenidos:

Introduccion	3
Procesadores Paralelos: SIMD	4
Procesadores Paralelos: MIMD	5
Procesadores Paralelos: Jerarquía y Taxonomía	6
Procesadores Paralelos: Clústers, I	7
Procesadores Paralelos: Clústers, II	8
Procesadores Paralelos: Algorítmica	9
Créditos, Bibliografía y Enlaces	10

Las opiniones vertidas en este artículo pertenecen únicamente a su autor, y no vinculan en modo alguno a Networking Center(R).

Este texto está protegido por las leyes de la propiedad intelectual. **Queda autorizada a todo individuo la reproducción para uso personal y sin objeto de lucro de este artículo, con la condición de que sea reproducido integralmente, incluida esta nota de copyright.**

Cualquier entidad o empresa que desee su reproducción debiera ponerse en contacto con el autor para solicitar su uso y llegar a un acuerdo mutuo sobre el mismo.

Networking Center(R) posee la autorizacion de uso como grupo de Usuarios sin animo de lucro.

Introducción

Desde que en 1981, Hockney Jesshope publicase su *Parallel Computers*, hasta nuestro días han pasado unos 20 años. En 20 años, el avance de la tecnología conforme a la demanda y a la necesidad de mejorar la productividad en los sistemas de procesamiento, ha sido realmente excelente, pero en muchos casos, no suficiente.

La posibilidad de aumentar el número de recursos y unidades dentro de los sistemas de información es una posibilidad viable, pero siempre poco rentable. Mejorar la cantidad física de memoria, intercambiar microprocesadores por otros de mejor tecnología y/o mayor frecuencia de trabajo de reloj, suele ser insuficiente.

Podríamos estar planteando numerosas opciones, pero siempre caemos en la duda: Y si necesitásemos que existiese en el sistema una mejora en el rendimiento de 50 veces el actual? Y si fuesen 100? Que mejor para ello, que valernos de la experiencia de otros, y utilizar el concepto de Ley de Amdahl: El posible aumento del rendimiento para una mejora dada esta limitado por la cantidad que se utiliza la característica mejorada.

No solo con aumentar una parte del módulo del computador es suficiente, e incluso, puede ralentizar nuestro tiempo de ejecución.

La posibilidad de implantar la técnica de procesamiento paralelo, es compleja, de lenta implantación en el mundo actual, pero muy escalable dados los avances actuales de la tecnología.

Paralelizar un proceso consiste en mejorar el tiempo de ejecución de un programa específico mediante la minimización de un problema en problemas más simples; Es decir: separar una tarea en sub tareas para que estas puedan tratarse por varios procesadores de forma paralela y simultánea.

Las técnicas de diseño de procesadores actuales, cuando más simples, mínimamente incorporan técnicas de mejora de productividad tales como la segmentación (pipelining), en las que las unidades funcionales locales de un microprocesador trabajan en diferentes sub tareas de una instrucción de forma simultánea, mejorando notablemente la frecuencia de ejecución de instrucciones. También disponemos de técnicas superescalares de procesamiento, en las que se dispone de varios cauces (pipelines) para la ejecución múltiple y segmentada de varias instrucciones simultáneas.

Podríamos enumerar bastantes diseños como los Alpha 21264, con un CPI (Ciclos por Instrucción) de 1/4 (Ejecuta 4 Instrucciones por ciclo de reloj) o procesadores de tipo vectorial, capaces de operar como un map (Véase Operaciones sobre Arrays, ManPage: `perlfunc`) sobre los elementos de un vector dado.

A partir de ahora, comenzaremos a tratar el tema de trabajo con multiprocesadores, varios procesadores ejecutando instrucciones simultáneamente y de forma distribuida.

Procesadores Paralelos: SIMD

En 1966, Flynn propuso un sencillo modelo de organización de computadores que aun en la actualidad se utiliza conforme fue propuesto. Flynn escrutó los componentes mas decisivos de la máquina, contando el numero de instrucciones paralelas y su flujo de datos, clasificando a posteriori los sistemas informáticos de la siguiente forma:

- A. **SISD**: Single Instruction; Single Data
- B. **SIMD**: Single Instruction; Multiple Data
- C. **MISD**: Multiple Instruction; Single Data
- D. **MIMD**: Multiple Instruction; Multiple Data

Las máquinas utilizadas por Flynn fueron los IBM 740 Y 7090, modelos SISD de la época; Hoy en día podríamos utilizar los MIPS R2000 y R3000 como modelos SISD. Proponemos pues como arquitecturas para este tema, los SIMD y los MIMD.

Los computadores SIMD operan perfectamente sobre arrays (vectores de datos). SIMD posee multiples unidades funcionales para paralelizar el trabajo. Su virtud es que todas estas unidades funcionales de ejecución en paralelo están sincronizadas y todas responden adecuadamente a una sola instrucción que proviene de un único contador de programa (Véase Estructura de un Camino de Datos).

Así pues, cada unidad de ejecución posee su propio grupo de recursos, es decir, propios registros de direcciones, lo cual permite tener diferentes direcciones para datos.

El desarrollo de SIMD viene motivado por el reducido espacio de memoria que necesita, dado que solo requiere la zona de programa en ejecución en memoria (El concepto de localidad espacial aparece ligado a SIMD desde su origen). Los métodos de intercambio de datos en SIMD se basan en redes de interconexión.

Para aprovechar un sistema SIMD, deberemos pensar en ejecución de bucles sobre arrays, por ejemplo, bucles FOR. Los sistemas SIMD han quedado en plano de investigación o estudio dada su poca flexibilidad.

Como ejemplos de maquinas SIMD disponemos de la serie CM de Thinking Machines, que pueden incluir hasta 65.535 procesadores trabajando a 7 Mhz de frecuencia y máximo de 512 Mbs de RAM (1987). Maspar también dispone de los MP-1216 con frecuencias de 25 Mhz y máximo de 1024 Mbs de RAM (1989).

Si analizamos las maquinas de tipo MIMD, encontraremos una idea de diseño bastante utópica; Es el llamado “ El Dorado “ de los computadores: Diseñar computadores potentes mediante la conexión de otros de menor tamaño.

Procesadores Paralelos: MIMD

La base de la idea MIMD es asociar tantos procesadores como se pueda o desee y obtener un rendimiento proporcional a ello. Los MIMD son también máquinas que ofrecen escalabilidad; Esto es, tanto hardware como software permiten un número variable de procesadores.

Gracias al diseño de software escalable, los MIMD pueden soportar pérdidas de procesadores por fallo, es decir, si un procesador falla en un sistema de N procesadores, el sistema obtendrá como resultado a ese fallo un servicio de $(N - 1)$ procesadores.

Los MIMD definen un alto rendimiento y una alta productividad (Esto es una de las relaciones más difíciles del diseño hardware), y esto contrasta al ejecutar una única tarea en procesadores múltiples. Los MIMD pueden tener el rendimiento más absoluto.

Tenemos múltiples procesadores, y ahora se nos plantea el problema de los accesos a memoria. La solución de MIMD es doble, bien utilizaremos memoria compartida, permitiendo que todos los procesadores accedan al mismo espacio de direcciones e implementaremos técnicas de control y protección de datos, bien utilizaremos memorias distribuidas, de forma que cada procesador tenga su propia memoria independiente.

Si nos decantamos por la idea de una memoria compartida, por múltiples procesadores, estaremos ante un submodelo de la clase MIMD, como todos los que se tratan a continuación. El modelo SMP o Symmetric Multi-Processing.

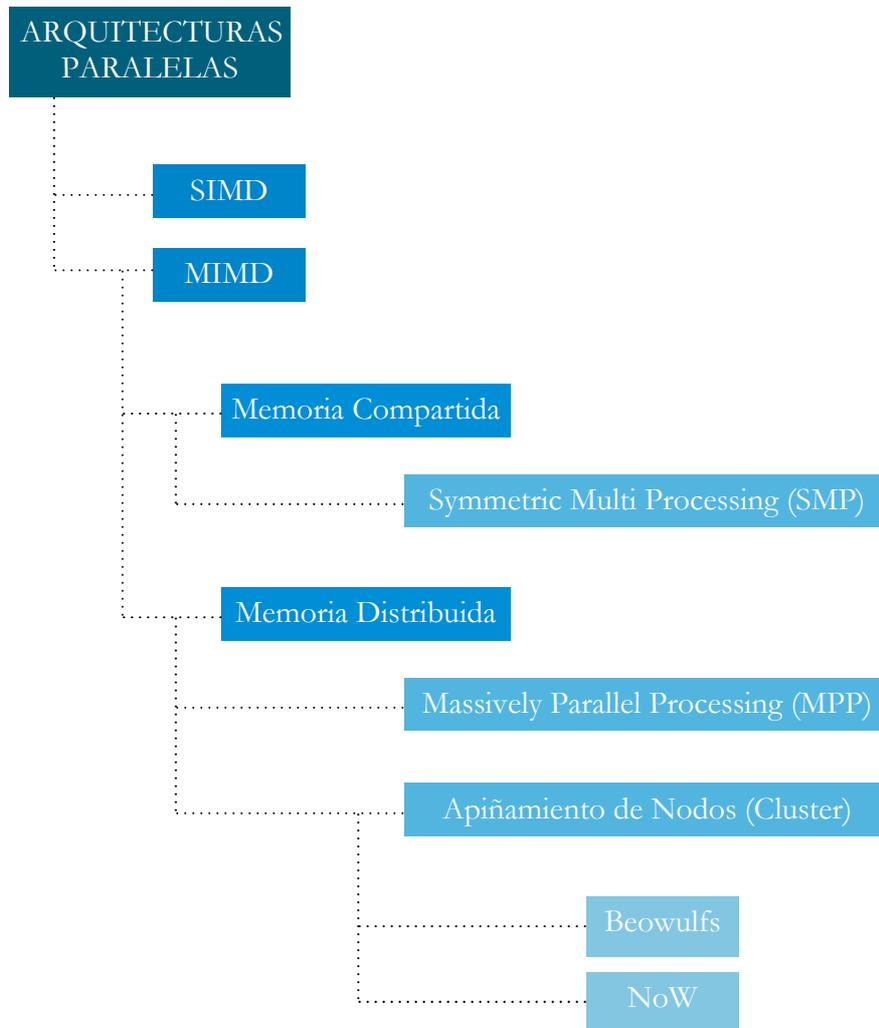
SMP hoy en día ha podido llegar muchas veces a los hogares gracias a la Intel MPS (Intel Multi Processing Specification), de forma que se permite trabajar con dos microprocesadores Intel en el mismo computador. Un sistema así configurado podría ejecutar tareas en diferentes procesadores Intel Pentium (Normalmente versiones III Xeon) sin interacción del usuario para ello.

En la asociación de SMP con Linux, diremos que es perfectamente posible tras una recompilación de núcleo, indicando que queremos realizar SMP. Para aquellos realmente interesados en el estudio de SMP bajo Linux, se recomienda la lectura de “Linux Parallel Processing HowTo”.

Tomando la filosofía de utilizar una memoria distribuida, nuestra jerarquía de procesamiento paralelo crece para darnos a conocer dos nuevos tipos de computadores o supercomputadores. Los computadores MPP o Massively Parallel Processing son aquellos cuyo índice de procesadores supera los 100 en el peor de los casos. Como bien plantea John Hennesy, es un término muy utilizado, pero muy vagamente definido. Hennesy establece los 100 microprocesadores como línea divisoria entre computadores MPP y aquellos que no lo son.

Ejemplos de MPPs puede ser Linux/AP+, basado en un Fujitsu AP1000+ con 64 procesadores SuperSPARC, con unidades de disco específicas y controladores de red especiales. Los SGI T3E e IBM SP2 también son computadores MPP.

Procesadores Paralelos: Jerarquía



Jerarquía de Arquitecturas Multiprocesador

Procesamiento: Clústers

Como otro elemento de la jerarquía MD-MIMD (Memory Distributed MIMD) tenemos la parte que realmente nos interesa en nuestro planteamiento: los apiñamientos o clusters.

La analogía de los nodos de procesamiento paralelo y otras workstations permite pensar que podemos utilizar computadores de ámbito comercial para realizar computadores de mayor volumen de procesamiento, y como no, paralelo.

Utilizando redes LAN y conectando estaciones de trabajo a través de un hub central con gran ancho de banda, y teniendo en cuenta que la política de memoria distribuida de message sending (envío de mensajes) para el arbitraje y sincronización de los sistemas paralelos se minimice, podremos idear un sistema apiñado (cluster) tal que sus diferencias con un supercomputador paralelo sean ínfimas.

Los clusters aprovechan al máximo la filosofía MIMD. Cada una de las máquinas de un clúster puede ser perfectamente una estación de trabajo, cuyo enfoque no es simplemente la dedicación completa al clúster; Esto es, puede ser aprovechada para otros menesteres cuando no se use como miembro del clúster. Podríamos incluso sacar rendimiento a todo el conjunto de ciclos muertos de un CPU mientras este realiza otras tareas no relativas a procesamiento paralelo en el clúster.

La relación calidad / precio es excelente. Téngase en cuenta que los componentes de un clúster son estaciones del día a día, lo cual es relativamente menos costoso que cualquier sistema de la misma rama (Léase MPP). Las redes de interconexión, en este caso, de área local, cada vez son más rápidas y menos costosas en el campo económico, luego cada vez tenemos más puntos a nuestros favor: Calidad / Precio magnífico. Aún así, el ancho de banda y la latencia del sistema es baja si lo comparamos con un sistema SMP (Y esto suponiendo que el Cluster esté completamente dedicado y aislado en un entorno de red).

La escalabilidad de MIMD aparece patente en los Clusters (Todo miembro inferior de la jerarquía hereda las características generales de su predecesor). Mientras que encontrar computadores MPP de más de 100 procesadores, es tarea difícil de encontrar (Difícil, no imposible. Véase si no el CM-5 de Thinking Machines Corp. Con 1.204 Procesadores a 33 Mhz y 4.096 unidades de punto flotante), y ver computadores SMP con más de 4 procesadores también lo es, en lo que se refiere a Clústers, veremos grupos de cómo mínimo 16 estaciones. La construcción de un Clúster con centenas e incluso unidades de mil de estaciones no sería un trabajo excesivamente complejo comparándolo con otros tipos de computadores paralelos.

Los clústers heredan de MIMD la posibilidad de proveer un servicio proporcional al número de unidades que se hallen trabajando (escalabilidad Hardware). Como vimos anteriormente, el fallo de una unidad de procesamiento, producirá un servicio de una unidad inferior al servicio general, sin provocar la caída del sistema completo.

Procesamiento: Clústers

A nivel de programación de un clúster, podemos decir que el nivel de estandarización actual permite asegurar la portabilidad de código entre un clúster y otro. De cualquier modo, existen contadas versiones de software que trabajen adecuadamente en un clúster. Cualquier programa UNIX como “time” o la propia función `time()` darán información relativa a la ejecución de un proceso en funciones de tiempo transcurrido, de sistema de usuario, y de CPU del nodo local, pero nunca a nivel general del cluster. Otro ejemplo interesante sería “ps”, que nunca dará procesos para todo el clúster. Aunque todo esto, parece ir cambiando conforme el paso de los años :-)

Hablaremos de dos tipos de Clusters: Los Beowulf, un cluster en el que todos los nodos son dedicados, y los NoW, o Network of Workstations, en el que se da otro uso, como ya comentamos en paginas anteriores, a las estaciones, no solo a nivel de cluster, sino también como estaciones de trabajo en algún momento.

Beowulf es un mito escandinavo que aparece en el primer texto fundamentado en Ingles; Se narran sus aventuras y proezas. El termino Beowulf fue otorgado a este diseño MIMD por la National Aeronautics and Space Agency de los EEUU y su intención inicial fue el desarrollo de un supercomputador capaz de batir records en lo referente a medidas de rendimiento: Se pretendía escalar a 1 GigaFlop, o lo que es lo mismo, realizar mil millones de instrucciones de tipo coma flotante por segundo. (Como comentario decir que los *flop al igual los *MIPS son ambiguos en lo relativo a medidas de rendimiento, y que no nos sirven como método comparativo entre maquinas de diferente arquitectura. Por ello, se recomienda la lectura de Análisis de Rendimiento de la bibliografía que se especifica en la sección correspondiente).

El primer Beowulf que vio la luz, en 1994, obtuvo un rendimiento de 1,25 gbFlops, lo cual demostraba que con material de bajo coste, se podía obtener un gran núcleo de procesamiento. Este primer Beowulf estaba sostenido por un Linux, y basado en 16 procesadores i486 DX4 a 100Mhz. Actualmente disponemos de los 11 GigaFlops de Naegling, basado en 140 microprocesadores Pentium Pro (i686).

La ventajas de Beowulf sobre la arquitectura NoW estan basadas principalmente en la facilidad para mantener un equilibrio de tareas entre todos los procesadores del Bw. Esto se debe a una menor latencia y a la mejor arbitración de transito de datos a traves de la red de arrea local, debido a que no existe nada mas en ejecución a traves de la misma. Se puede aumentar el rendimiento de un Bw mediante la parametrización del mismo, para que se adapte en mejores condiciones a una aplicación paralela que se vaya a ejecutar.

El problema de los clusters se basa principalmente en la red local. Esta redes, normalmente de tipo Ethernet de 10 Mb/s constituyen un cuello de botella en la técnica de message sending (o envío de mensajes; véase técnicas de control y arbitraje en sistemas multiprocesador). Según comenta Donald Becker, uno de los principales miembros del proyecto, considero útil incluir una segunda interfaz de red, para separar procesos de arbitraje e incremento así en un 70% el rendimiento del cluster.

Procesamiento: Algorítmica

El cuando y porque utilizar un sistema de procesamiento paralelo nace de la necesidad de acelerar la ejecución de un problema y su algoritmo específico. No solo es orden de la resolución del mismo el computador, también la magnitud y dificultad del problema serán participes en el incremento o decremento del tiempo de ejecución. Podemos utilizar el concepto de Amdhal para obtener estas conclusiones.

Así mismo, el fragmentar el problema en otros de menor magnitud, permitirá mejorar en un computador paralelo la ejecución, dado que podremos asignar cada fragmento a un procesador específico, y permitir que este opere con sus datos, aquellos que le corresponde, sin la necesidad de estar ocupando a sus compañeros en la tarea adjudicada. En lo referente a fragmentar el control, un entorno de objetos distribuidos, tal como es CORBA, será mucho mas potente y adecuado para nuestra solución.

Se dice que obtenemos aceleración lineal cuando tratamos de solucionar nuestro problema con X procesadores, siendo la aceleración X veces mas rápida que con un solo procesador. Los algoritmos que se ejecutan sobre computadores paralelos suelen ser de carácter básico y con bastante II (Information Interchange) entre los procesadores miembros del macro computador.

En otros casos, se necesita que cada procesador miembro tenga acceso a conocer cual es el estado actual del computador completo, es decir, de todo el sistema. Esto supone grandes gastos, debido a que la información debe ser de todos para todos, y su correcta propagación es realmente compleja y costosa.

Planteamos como correcta la ejecución de una aplicación en un cluster si y solo si esta ha sido suficientemente paralelizada, y para ello es necesario recurrir a la programación concurrente. Existen dos métodos de paralelizar: Utilizando herramientas propias para la programación del envío de mensajes interprocesadoral o bien, haciéndolo a mano. Esta herramientas, los paralelizadores son excesivamente costosos y producen código paralelo poco eficiente. Luego esta opción es mas bien desestimable. Aun así, se continua desarrollando gran cantidad de estos programas y mejorándolos día a día.

La idea de programarlo a mano utilizando un lenguaje de programación paralelo tiende a estar mas cerca del programador. Los lenguajes mas frecuentes en este campo con Fortran de Alto Rendimiento o HPF, Fortran 95 que incluye como estándar desde 1997 la mayoría de mejoras de HPF sobre Fortran 90, y C. El hecho de programar el paso de mensajes nosotros nos permite aprovechar al máximo el tiempo, pero siempre tiene posibilidad de errores y reduce la portabilidad entre maquinas del mismo tipo.

Por ello, lo mas recomendable es el uso de librerías que permitan al programador desarrollar el mensaje entre dos procesadores o a nivel mas masivo. **PVM**, o Parallel Virtual Machine, y **MPI**, o Message Passing Interface son los casos mas significativos para ello.

Procesamiento: Créditos

- Bibliografía:

- **Organización y Diseño de Computadores.** Hennesy-Patteron. Ed. McGraw Hill.
- **Computers Architecture: A quantitative Approach.** Hennesy-Patteron. E McGraw Hill
- **Fundamentos del Diseño Digital.** Thomas L. Floyd. Ed. Prentice Hall.
(Aplicaciones al Diseño de Caminos de Datos y Unidades de Control. Diseño MonoProc)

- Enlaces en la Web:

- **Linux Parallel Processing HowTo:** <http://www.linuxdoc.org>
- **Extreme Linux:** <http://www.redhat.com/extreme>
- **Top500:** <http://www.top500.org>
- **HPF:** <http://www.mhpcc.edu/doc/hpf/hpf.html>
- **Fortran 95:** <http://www.nsc.liu.se/f95.html>

- Agradecimientos:

- **D. Pedro Medina,** Catedrático de la Univ. de Las Palmas de Gran Canaria.

Gracias por haberme introducido al mundo del desarrollo de microprocesadores de una forma tan amena :-)

- **Gema Gómez.**

Gracias por tu apoyo y por aguantar mas de una vez mis conversaciones :-)
Y también por ayudarme en la correccion de este documento :-)