**Reporte**

**Multiprocesadores**

Alvaro Izaguirre Serrano

Adrián García Betancourt

**Instituto Tecnológico y de Estudios Superiores de Monterrey**

**Maestría en Ciencias Computacionales**

**Programación Paralela**

Fecha de Entrega: 29/agosto/2014

**Definición**

Se denomina multiprocesador a un computador que cuenta con dos o más [microprocesadores](http://es.wikipedia.org/wiki/Microprocesador) ([CPUs](http://es.wikipedia.org/wiki/CPU)).

Gracias a esto, el multiprocesador puede ejecutar simultáneamente varios [hilos](http://es.wikipedia.org/wiki/Hilo_de_ejecuci%C3%B3n) pertenecientes a un mismo [proceso](http://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29) o bien a procesos diferentes.

Los ordenadores multiprocesador presentan problemas de diseño que no se encuentran en ordenadores monoprocesador. Estos problemas derivan del hecho de que dos [programas](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29) pueden ejecutarse simultáneamente y, potencialmente, pueden interferirse entre sí. Concretamente, en lo que se refiere a las lecturas y escrituras en memoria. Existen dos arquitecturas que resuelven estos problemas:

* La arquitectura [NUMA](http://es.wikipedia.org/wiki/NUMA), donde cada procesador tiene acceso y control exclusivo a una parte de la memoria.
* La arquitectura [SMP](http://es.wikipedia.org/wiki/SMP), donde todos los procesadores comparten toda la memoria.

Esta última debe lidiar con el problema de la coherencia de caché. Cada microprocesador cuenta con su propia [memoria cache](http://es.wikipedia.org/wiki/Memoria_cache) local. De manera que cuando un microprocesador escribe en una dirección de memoria, lo hace únicamente sobre su copia local en caché. Si otro microprocesador tiene almacenada la misma dirección de memoria en su caché, resultará que trabaja con una copia obsoleta del dato almacenado.

Para que un multiprocesador opere correctamente necesita un [sistema operativo](http://es.wikipedia.org/wiki/Sistema_operativo) especialmente diseñado para ello. La mayoría de los sistemas operativos actuales poseen esta capacidad.

**Requerimientos**

Para un sistema multiprocesador es necesario contar con los siguientes requerimientos:

* Gran espacio de almacenamiento, el cual sea accesible por múltiples rutas.
* Gran disponibilidad, componentes tolerantes a fallas y soporte del sistema operativo.
* Independencia del procesador, se debe de permitir cualquier operación en cualquiera de los procesadores, sistema de archivos globales, balance de carga y que el sistema pueda ser escalable.

**Taxonomía**

Los multiprocesadores se basan principalmente en dos diseños clusters y multiprocesadores de memoria compartida, este reporte se basa en los multiprocesadores de memoria compartida.



**Jerarquías de Memoria**

Una forma de reducir la latencia es reduciendo la cantidad de estos accesos a la memoria por parte de los procesadores, y se puede conseguir incorporando jerarquías de memoria en los mismos. De esta forma, si los programas explotan la localidad de datos, los accesos a memoria se reducirán.

Las máquinas UMA se hicieron más y más populares desde aproximadamente el año 2000, año en el que podríamos decir que hubo un punto de inflexión en la tendencia de los diseños de las arquitecturas de los uniprocesadores, debido básicamente al consumo energético y a la consecuente bajada de rendimiento en las aplicaciones. La aparición de los procesadores CMP o chip multiprocesador fueron el resultado natural de intentar aprovechar las mejoras tecnológicas existentes, y aprovechar así mejor el área del chip, sin aumentar considerablemente el consumo energético. Los procesadores CMP son procesadores de tipo UMA que incorporan más de una unidad de procesamiento dentro de un único chip.

Los nombres que reciben las arquitecturas multiprocesador UMA y NUMA tienen que ver con el tiempo de acceso a la memoria principal, no teniéndose en cuenta la diferencia de tiempo entre un acierto o un fallo en caché. De lo contrario, también deberíamos considerar arquitectura NUMA a cualquier sistema con jerarquía de memoria, incluidos los uniprocesadores.

En cuanto a la arquitectura COMA, que no tuvo mucho éxito, se basa en tener la memoria compartida como si fuera una gran caché.

**UMA**

En este tipo de arquitectura, como bien dice su nombre, todos los accesos a memoria tardan el mismo tiempo. Seguramente podemos pensar que es difícil que tengamos el mismo tiempo de acceso si la memoria, aunque compartida, está dividida en módulos a los que se accede a través de una red de interconexión basada en switches. Eso es cierto y, para conseguir esta uniformidad de acceso, se tiene que aumentar el tiempo de los accesos más rápidos.

¿Por qué se busca esta uniformidad de tiempo de acceso? Porque para los programadores es más fácil deducir qué parámetros ayudarán a mejorar sus programas si el tiempo de acceso es igual para cualquier acceso. De lo contrario, como pasa con las arquitecturas NUMA, deberían ser mucho más conscientes de la arquitectura que tiene el computador y de cómo están distribuidos los datos en la arquitectura en cuestión.

Este tipo de máquinas son fáciles de construir. Sin embargo, el hecho de que todos los procesadores accedan al mismo bus para acceder a memoria puede significar un cuello de botella. Una alternativa a tener un único bus sería tener varios buses, de tal forma que se distribuirían los accesos a los módulos de memoria. También se podría usar una red de interconexión más compleja, como una red crossbar, que permita evitar conflictos entre los accesos a las memorias que no van al mismo módulo.

* Multiprocesadores basados en comunicación de bus, comparte la memoria principales, además de poder contar con memoria cache en cada procesador además de memoria privada.

|  |  |
| --- | --- |
|  | La latencia del bus limita el número de procesadores que pueden ser interconectados. |
|  | Menor latencia, sin embargo se requiere de mecanismos para mantener la coherencia de las memorias cache, lo cual incrementa la complejidad de las aplicaciones. |
|  | El compilador tiene la capacidad de guardar datos en la memoria privada, requiere de un diseño mas detallado. |

Ventajas:

* Es el más sencillo de conectar
* Mejor escala de rendimiento desde el punto de vista económico con el número de nodos/procesadores conectados al bus.
* Todos los procesadores están conectados de forma directa con la memoria de forma directa.
* Todos los procesadores están a distancia de un bus de la memoria.

Desventajas:

* Poca escalabilidad
* Incremento de latencia cuando el número de procesadores aumenta
* Coherencia de cache cuando se cuenta con este tipo de memoria
* Multiprocesadores en configuración crossbar - Esta consiste en una malla de switches que permite la conexión no bloqueante de cualquier procesador a cualquier banco de memoria. Cada switch del crosbar puede ser electrónicamente cerrado o abierto, permitiendo o no que la línea vertical se conecte a la línea horizontal, respectivamente.

|  |
| --- |
| 8-3 |
| * Soporta un mayor número de procesadores * Es una red que evita bloqueos * Es una configuración que mantiene una buena relación costo/desempeño en configuraciones de hasta 100 procesadores – teniendo una tasa de crecimiento . |

¿Qué significa que la malla de switches permite una conexión no bloqueante? Significa que el acceso de un procesador a un banco de memoria no bloquea la conexión de otro procesador a cualquier otro banco de memoria. Es por eso mismo por lo que normalmente el número de bancos de memoria es mayor que el número de procesadores. En otro caso, habría procesadores que no se podrían conectar a un banco de memoria debido a un conflicto para acceder al mismo banco que otro procesador.

La desventaja de este sistema de conexión con memoria es que se precisan alrededor de switches para poder montar este sistema, siendo este coste importante y, por consiguiente, poco escalable. En cambio, un sistema basado en un bus es escalable en términos de coste, pero no de rendimiento.

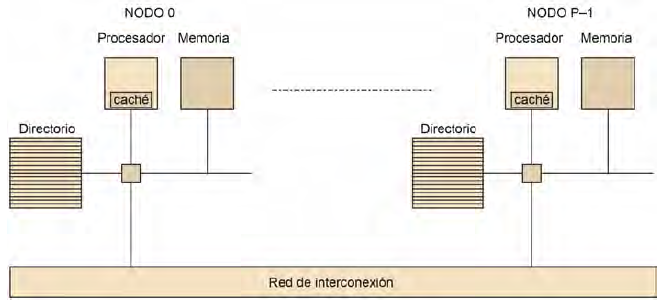
**Ejemplos de Fabricantes**

* Intel Core2 Duo (con 2 CPU) - Homogéneo
* Intel Nehalemi7 (con 4 CPU) - Homogéneo
* AMD Istanbul (con 6 CPU) - Homogéneo
* Sun SPARC64-VIIIfx (con 8 CPU) - Homogéneo
* IBM Power7 (con 8 CPU) - Homogéneo
* Intel Sandy Bridge, que tiene un procesador de carácter general y una unidad de procesamiento gráfico - Heterogéneo

**NUMA**

En los multiprocesadores NUMA, a diferencia de los UMA, los accesos a memoria pueden tener tiempos distintos. En estas máquinas la memoria también está compartida, pero los módulos de memoria están distribuidos entre los diferentes procesadores con el objetivo de reducir la latencia.

El módulo de memoria que está junto a un procesador en un mismo nodo recibe el nombre de memoria local a este procesador. Así, los accesos de un procesador a su memoria local suelen ser mucho más rápidos que los accesos a la memoria local de otro procesador (memoria remota). Las redes de interconexión típicas de estos sistemas son las redes de tipo tree y las de bus jerárquico.



Por otra parte, al igual que pasaba con las arquitecturas UMA, también se incorporaron las cachés para reducir la contención de memoria. Pero además, con estas cachés también se intenta ocultar la diferencia de tiempo de acceso a memoria entre memoria local y remota.

Según tengan o no caché, los multiprocesadores NUMA se clasifican en cache coherente NUMA (ccNUMA) o No-Coherente NUMA respectivamente. El término de coherencia o no coherencia proviene de mantenerla o no en los datos duplicados en las cachés.

**Ejemplos de Fabricantes**

* SGI Altix UV 1000, que está formado por 224 procesadores Intel Xeon X7542 de 64 bits, a 2,66 GHz de 6 núcleos, con un total de 1.344 núcleos de cálculo
* SGI Origin 3000
* Cray T3E
* AMD Opteron.

**COMA**

Aunque las arquitecturas ccNUMA ayudan a ocultar la diferencia de tiempo entre accesos locales y remotos, esto dependenderá de la localidad de datos que se explote y la capacidad de las cachés. Así, en el caso de que el volumen de los datos a los que se accede sea mayor que la capacidad de la caché, o el patrón de acceso no ayude, volveremos a tener fallos de caché y el rendimiento de las aplicaciones no será bueno.

Las arquitecturas COMA (cache only memory access) intentan solucionar este problema haciendo que la memorial local a cada procesador se convierta en parte de una memoria caché grande. En este tipo de arquitecturas, las páginas de memoria desaparecen y todos los datos se tratan como líneas de caché. Sin embargo, con esta estrategia surgen nuevos interrogantes: ¿cómo se localizan las líneas de caché? y cuando una línea se tiene que reemplazar, ¿qué sucede si esta es la última? Estas preguntas no tienen fácil solución y pueden requerir de soporte hardware adicional. Normalmente se implementan utilizando un mecanismo de directorio distribuido.

**Ejemplos de Fabricantes**

* KSR-1
* Data diffusion machine

**Consistencia de Memoria**

El problema de consistencia de memoria aparece cuando una memoria está formada por varios módulos de memoria conectados a través de una red de interconexión a un conjunto de procesadores. En particular, cuando varios procesadores acceden a un dato para escritura se nos plantea la pregunta siguiente: ¿en qué orden se deben ver estas escrituras por parte del resto de procesadores si éstos hacen una lectura de los datos de forma paralela?

Hay muchas respuestas posibles, ya que unos accesos se pueden adelantar a los otros, siendo el resultado de las lecturas impredecible. Todo esto se agrava cuando tenemos cachés, ya que entonces es posible que varios procesadores tengan copias de lecturas previas.

Para que los procesadores que acceden a unos datos sepan, por adelantado, qué valores pueden obtener si hay escrituras y lecturas a la vez, se suelen determinar unas reglas que determinan lo que puede devolver la memoria en esa situación. El conjunto de estas normas es lo que recibe el nombre de modelo de consistencia. Por ejemplo, si una CPU0 realiza un acceso de escritura del valor X en una posición de memoria, la CPU1 realiza también un acceso de escritura del valor Y en la misma posición de memoria, y la CPU2 realiza un acceso de lectura del valor de esa misma posición de memoria, ¿qué debería obtener CPU2? Puede ser que obtuviera el valor X y no el valor Y, siendo esto posiblemente correcto si el modelo de consistencia, que todos los procesadores deben conocer, permite este resultado.

**Consistencia Estricta**

En este modelo de consistencia, cualquier lectura sobre una posición de memoria deberá tener como resultado la última escritura realizada en esa posición de memoria.

Esta consistencia se puede alcanzar si hay un único módulo de memoria que trate en estricto orden de llegada los accesos, y sin ningún tipo de caché en el sistema. Esto se puede hacer pero se traducirá en que el acceso a memoria será un cuello de botella.

**Consistencia Secuencial**

La consistencia secuencial garantiza que los valores que se ven en memoria se vean en el mismo orden por todas las CPU. No importan si es en un orden diferente al que se hicieron las escrituras. Con este modelo no es posible que dos CPU, que leen un dato de una posición de memoria varias veces al mismo tiempo que otras CPU están escribiendo sobre la misma posición, vean dos secuencias diferentes de valores en esa posición de memoria. El orden en el que se ven las escrituras bajo este modelo no es importante, ya que el que se produzca antes una escritura u otra será aleatorio. Lo importante es que cuando se establezca este orden, todas las CPU vean el mismo orden.

**Consistencia del Procesador**

El modelo de consistencia del procesador indica que las escrituras hechas por un procesador sobre una posición de memoria serán vistas por los demás procesadores en el mismo orden en el que se lanzaron. Sin embargo, a diferencia del modelo de consistencia secuencial, no garantiza que los accesos de escrituras de diferentes procesadores se vean igual por los demás procesadores.

**Consistencia Débil**

La consistencia débil no garantiza ningún orden entre las escrituras de un mismo procesador, pero dispone que haya unas operaciones de sincronización que garantizan que todas las operaciones de escrituras se deben acabar antes de que esta operación acabe.

Las operaciones de sincronización vacían el pipeline de escrituras, y siguen un modelo consistente secuencial, de tal forma que los procesadores ven un mismo orden en las operaciones de sincronización. Estas operaciones se deben realizar a nivel software y no son a costo cero.

**Consistencia Release**

El modelo de consistencia release intenta reducir el costo de esperar todas las escrituras previas a una operación de sincronización. Este modelo se basa en la idea de las secciones críticas. La filosofía es que un proceso que sale de una sección crítica no necesita esperar que todas las operaciones de escritura previas se hayan acabado. Lo que sí que es necesario es que si otro proceso (o éste mismo) tiene que entrar en la sección crítica, se tendrá que esperar a que las operaciones previas a la salida de la sección crítica sí que hayan acabado. De esta forma se evita tener que parar los accesos hasta que realmente sea necesario.

**Coherencia de Cache**

El problema de consistencia de memoria aparece cuando una memoria está formada por varios módulos de memoria conectados a través de una red de interconexión a un conjunto de procesadores. En particular, cuando varios procesadores acceden a un dato para escritura se nos plantea la pregunta siguiente: ¿en qué orden se deben ver estas escrituras por parte del resto de procesadores si éstos hacen una lectura de los datos de forma paralela?

Esto no es un problema si no hay ninguna actualización del dato. En cambio, si se produce una escritura en el dato por parte de un procesador, el resto de copias del valor, en el resto de cachés, quedarán desactualizadas. En este caso hay un problema de coherencia de la caché.



**Protocolos de Escritura**

Para mantener la coherencia de caché en los sistemas multiprocesador se pueden seguir dos políticas en la escritura de un dato:

1. Write-updated o write broadcast: Mediante este protocolo se mantienen actualizadas las copias del dato (bloque de memoria) en las cachés del resto de procesadores.
2. Write-invalidate: Mediante este protocolo se pretende asegurar que el que escribe tiene la exclusividad del dato (bloque de memoria), haciendo que el resto de procesadores invaliden su copia y no puedan tener copias desactualizadas. El dato (bloque de memoria) es actualizado en memoria cuando es flushed/reemplazado de la caché o lo pide otro procesador. Con este protocolo, el dato (línea de caché con el bloque de memoria donde está el dato) tiene unos bits de estado para indicar exclusividad, propiedad y si ha sido modificado (dirty).

La segunda política es la que se suele usar más, ya que normalmente tiene mejor rendimiento que la primera.

**Mecanismo Hardware**

La forma de mantener la coherencia de caché en hardware es mediante la utilización de bits de estado para cualquier bloque de memoria que se comparte. Hay dos mecanismos hardware que mantienen actualizado el estado de los datos compartidos:

* Sistema basado en Snoopy: los bits de estado del bloque de memoria están replicados en las cachés del sistema que tienen una copia del dato (bits por cada línea de caché), a diferencia del basado en directorios, que está centralizado. Son sistemas en los que los procesadores suelen estar conectados a la memoria vía bus, y además, cada controlador de caché tiene un hardware dedicado que puede leer/sondear (Snoopy) qué pasa por el bus. Así, todos los controladores de memoria miran si las peticiones de accesos que se realizan sobre el bus afectan a alguna de sus copias o no.
* Sistema basado en directorios: Los bits de estado de compartición de un bloque de memoria están únicamente en un sitio llamado directorio. Cuando se realizan lecturas y escrituras sobre un dato del bloque de memoria, su estado, centralizado en el directorio, se deberá actualizar de forma adecuada.

**Protocolos de Coherencia**

Existen dos protocolos de coherencia (MSI y MESI) que se pueden combinar con la política en escritura write-invalidate y los mecanismos de hardware mencionados. Estos protocolos determinan los estados de compartición de memoria en los que puede estar un bloque de memoria, y cómo se realiza la transición de uno a otro según los accesos que se produzcan en el sistema multiprocesador.

**MSI**

El protocolo MSI de coherencia contempla los siguientes estados:

1. Dirty o modified (M): El estado en el que un bloque de memoria (línea de caché) está en la caché cuando se ha modificado o se ha leído para modificar.
2. Shared (S): El estado en el que un bloque de memoria está cuando se ha leído y no hay intención de modificarlo.
3. Invalid (I): El estado en el que un bloque de memoria (línea de caché) está cuando se ha invalidado o bien no existe en la caché.

¿Cómo aplicaríamos estos estados a una máquina con política en escritura write-invalidate y un sistema hardware basado en bus (Snoopy)? El grafo de estados determina los estados en los que puede estar una línea de caché (nodos del grafo) y las acciones del bus o del procesador que provocan los cambios de un estado al otro (aristas del grafo). Las aristas del grafo están etiquetadas con el siguiente formato: <origen de la acción / transacción en el bus realizada como parte de la acción>. Las operaciones o acciones que se pueden realizar por parte del procesador son : PrRd (el procesador hace una lectura Rd) y PrWr (el procesador realiza una escritura Wr). Las acciones que se pueden realizar a través del bus son: BusRd (se realiza un petición de lectura a memoria), BusRdX (se realiza una petición de lectura con intención de escritura), y flush se deja en el bus el bloque de memoria que contiene el dato y, si se da el caso, la caché del procesador que la solicitó, puedan actualizar la línea. Estas acciones en el bus se ven en el resto de procesadores gracias a los mecanismos hardware para mantener la coherencia. Por ejemplo, en un sistema basado en Snoopy, el SCC del sistema podrá ver qué pasa por el bus y actuar en consecuencia. (la línea de caché), para que la memoria.



**MESI**

El protocolo de coherencia MESI tiene un estado más que el protocolo MSI: el de exclusividad. Una línea de caché de un procesador está en este estado si el procesador realiza una lectura de un bloque de memoria y es el único que tiene copia de este bloque. Con este estado se intenta ahorrar operaciones en el bus, ya que al ser la única copia no es necesario informar al resto de procesadores. Por otro lado, a diferencia del protocolo MSI en el que la memoria siempre suministra el bloque de memoria si lo tiene actualizado, en la versión original del protocolo MESI (la versión de Illinois), la memoria sólo suministra un bloque de memoria si ésta es la única copia, independientemente de si está actualizado o no. Por el contrario, si hay una copia del bloque de memoria actualizada en una caché, será ésta la que suministre el bloque de memoria. Esto es lo que se llama la técnica cache-to-cache sharing, y su objetivo es suministrar más rápidamente los datos al bus. La técnica cache-to-cache sharing requiere que, en caso de que haya más de una caché con una copia del bloque de memoria actualizado, exista un mecanismo para determinar qué caché dará el bloque de memoria.

El grafo de estados, para un caso donde tenemos Snoopy y write-invalidate, y utilizando la técnica de cache-to-cache sharing de la versión original de Illinois, se muestra a continuación.



**Referencias**

* "Multiprocesadores y Multicomputadoras", Daniel Jimenez Gonzales, Universitat Oberta de Catalunya
* "Scheduling Real-TimeTasks in Multiprocessors and Distributed Syatems", Version 2 CSE, IIT Kharagpur
* "Distributed and Multiprocessor Scheduling", Steve J. Chapin, Syracuse University and Jon B. Weissman, University of Minnesota
* "An Introduction to Parallel Programming", Peter Pacheco, 1st Edition