IMPORTANCIA DE LA NO COHERENCIA DE CACHE EN UN SISTEMA MULTIPROCESADOR

Rafael B. García Jorge R. Ardenghi

Laboratorio de Investigación en Sistemas Distribuidos (LISiDi)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur – Bahía Blanca, Argentina
T.E.: +54 291–4595135 Fax: +54 291–4595136
{rbg,jra}@cs.uns.edu.ar

ABSTRACT

La importancia de la memoria cache en un sistema reduciendo el tiempo de acceso efectivo a memoria, se ve potenciada en los sistemas multiprocesador por su contribución a la escalabilidad. El modelo de consistencia de memoria mas aceptado es el secuencial SC, por ser el mas afín a la intuición del programador. Otros modelos, a partir de imponer determinadas características al programa, aseguran una ejecución secuencial con el beneficio de un manejo superior a nivel del software y del hardware. En nuestra opinión, un aspecto que incide fuertemente en la escalabilidad y/o complejidad de un sistema es el de coherencia de cache. Los modelos de consistencia secuencial, y muchos de los derivados de este último, requieren mantener coherente la cache. Esto obliga a una serialización de los accesos a locaciones individuales, de aplicación tanto en sistemas UMA (protocolos de *snooping*) como NUMA (generalmente esquemas de directorio). Dado que típicamente los programas son sincronizados, y en la hipótesis de que se debe ser mas eficiente en los casos mas frecuentes, analizaremos los resultados de diferir la coherencia hasta esos puntos, con el objetivo último de alcanzar acceso inmediato, *fast access*, y a su vez no obstaculizar la optimización a nivel de los compiladores.

Palabras Clave: Consistencia de memoria, Coherencia de Cache

1. INTRODUCCIÓN

Actualmente se puede observar una clara tendencia hacia los sistemas de memoria compartida, sustentado en la potencia de cómputo incremental y la facilidad de programación. Así podemos citar arquitecturas exitosas tanto UMA (*Uniform Memory Access*) como los servers SUN Enterprise, y de tipo NUMA (*Non Uniform Memory Access*) como la SGI Origin 2000/3000. Estos sistemas mantienen la coherencia de cache entre los procesadores individuales utilizando un protocolo de coherencia por hardware destinado a asegurar una consistencia secuencial. Esto constituye una limitación fundamental en cuanto a lograr un sistema eficiente y escalable. Las restricciones que impone van mas allá del orden parcial definido por las operaciones de sincronización en un pro-

grama paralelo, de lo que se deduce imponen un overhead innecesario.

En general para incrementar la concurrencia los sistemas multiprocesador soportan réplica de datos. Disponer de la misma variable en memoria local (cache) posibilita su uso simultáneo. Para garantizar la consistencia el sistema debe controlar dichas réplicas, bien sea por medio de invalidaciones o actualizaciones. Una propiedad interesante de los algoritmos que implementan un modelo de consistencia es cuánto tiempo toma una operación de memoria. Si dicha operación no requiere de que se termine ninguna comunicación, y por ende puede ser completada solo basada en el estado local del procesador del acceso, se dice que la operación es *fast*, lo cual ciertamente resulta un atributo deseable. Attiya y Welch [AF92] demostraron que ningún algoritmo secuencial puede garantizar una ejecución *fast* para todas sus operaciones.

Es central en la definición del modelo de consistencia SC, y de todos sus derivados, la coherencia de memoria, la que se debe entender como que "todas las escrituras a una misma locación son serializadas en algún orden el cual es válido para todos los procesadores". Obviamente constituye una forma menos restrictiva de serialización comparada con el requerimiento del modelo SC, dado que se refiere a locaciones individuales.

La semántica del modelo de programación paralelo típico se sustenta en un orden parcial para el programa y las operaciones en memoria, definido por la concurrencia y sincronización, y no en un orden total de las operaciones. Además la coherencia de cache representa un obstáculo adicional para la definición de un modelo de consistencia que sea entendido en todos los niveles, del software y del hardware [GS97].

El problema de coherencia de cache en sistemas multiprocesador ha sido abordado con soporte variado de hardware/software. Podemos sintetizar que los objetivos de interés son:

- El protocolo soporte de forma eficiente el *false sharing* de los bloques de datos.
- La solución aproveche las posibilidades de los modelos de consistencia *delayed*. Programas que hagan explícitas las operaciones de sincronización admiten modelos de consistencia mas relajados sin sacrificar la intuición de ejecución secuencial.
- Deslinde al programador y al compilador de la necesidad de considerar que el dato compartido está replicado en múltiples memorias locales.
- Se alcance una naturaleza local, *fast access*, para los accesos a memoria.

2. CONSISTENCIA EN LOS SISTEMAS MULTIPROCESADOR

Los sistemas actuales de memoria compartida están generalmente implementados por hardware. Esto significa que la coherencia entre los caches individuales se mantiene en forma automática y transparente. Dependiendo de la arquitectura base pueden distinguirse dos grupos: pasivos, basados en protocolos de snooping y activos, basados en algún tipo de directorio con información acerca del contenido de los caches remotos.

El representante mas conocido del primer grupo es el protocolo MESI o Illinois, típicamente

aplicado en los sistemas SMP (simétricos), basado en el bus, dado que generalmente dependen de que todos los procesadores o caches "escuchen" sobre el bus todo el tráfico de memoria. El problema con este protocolo es que al depender de esta propiedad tipo broadcast del bus hereda las limitaciones de los sistemas basados en un bus. Además, al requerir que todo el tráfico global sea propagado en un ciclo del bus, se limita la frecuencia del reloj del bus.

El segundo grupo, el de protocolos de coherencia activo, evita la dependencia de la observación de los eventos de coherencia sobre un bus común y por lo tanto habilita multiprocesadores con memoria compartida no basada en bus. Mantienen la información acerca de los datos replicados en los nodos remotos en forma distribuida. Se han planteado distintos métodos para manejar esta información, incluyendo directorios (Stanford DASH o la SGI Origin 2000/3000), o listas enlazadas mantenidas en hardware (Scalable Coherent Interface (SCI)). Sin embargo, sigue siendo necesario propagar todas las actualizaciones a los procesadores que tienen réplicas de los datos, provocando nuevamente comunicación global. Además, para el mantenimiento de los directorios o listas enlazadas, se requiere un hardware dedicado, posiblemente complejo, capaz de tener acceso directo al bus que comunica procesador y memoria.

3. IMPLICANCIAS DE UN MODELO RELAJADO

Como se analizó previamente una implementación típica de un sistema multiprocesador incluye coherencia de cache soportada por el hardware.Esto limita la escalabilidad del sistema subyacente, aumenta su complejidad, e impide una implementación con componentes comunes. Desde otro punto de vista, la coherencia se constituye en un obstáculo para obtener una definición del modelo de consistencia que abarque todos los niveles, *end-to-end*. La semántica del modelo de programación paralelo típico asume un orden parcial del programa y de las operaciones en memoria, lejos del orden total de la SC, el que resulta definido por la concurrencia y sincronización del programa.

3.1. Ventajas del Software

La coherencia agrega restricciones adicionales en el ordenamiento de las operaciones en memoria. Condiciona el ordenamiento en los accesos a memoria en diferentes subprogramas, de manera tal que se observen las escrituras en locaciones compartidas en un mismo orden. Esto dificulta la tarea del programador, el cual se verá precisado a introducir engorrosas modificaciones en el programa, por ejemplo insertando declaraciones "volatil" para determinadas variables.

Mas aún, muchas optimizaciones a nivel del compilador trabajan con representación del programa que reflejan un orden parcial, por ejemplo el algoritmo Sethi-Ullman de reordenamiento de instrucciones para mejorar la asignación de registros. Este y otros casos de optimización, sustentados en un reordenamiento de las instrucciones, resultan difíciles de compatibilizar con un modelo de consistencia que fije un ordenamiento en las instrucciones. Se podría conservativamente desabilitar todo reordenamiento de instrucciones, o de manera selectiva hasta que se alcance un comportamiento correcto. Un manejo mas integrador podrá eliminar la necesidad de estas modificaciones a nivel del programa, y por ende las eventuales

ineficiencias que se originan.

Se puede puntualizar además la distinción artificial que introduce la coherencia en cuanto al tamaño del dato, en tanto entiende con el tamaño fijado para el bloque de cache. Con respecto al orden total de las operaciones de memoria, que vimos no puede manejarse a nivel del programa fuente, podemos agregar que habrá operaciones en memoria en el código compilado que no resultan visibles a nivel fuente y aun más, a nivel del propio set de instrucciones de una arquitectura el orden de ejecución de los operandos de instrucciones compuestas podrá encontrarse no especificado.

3.2. Ventajas de la Arquitectura

La ventaja más significativa de las arquitecturas no-coherentes, comparadas a sus contrapartidas coherentes, es que pueden contar con módulos de procesador/cache completamente independientes. Se elimina la necesidad de cualquier componente o infraestructura global además de la de interconexión subyacente capaz de establecer la abstracción de una memoria global, la cual puede reducirse a la capacidad de lograr accesos remotos a memoria directamente desde el procesador local. Dado que esto no impone una complejidad significativa en el hardware se puede obtener un sistema con una buena escalabilidad, similar a la obtenida en sistemas distribuidos con pasaje de mensajes, pero con la ventaja de programar con memoria compartida.

En cuanto a una arquitectura moderna con *Out-of-Order Multi Issue*, el no requerir coherencia permite explotar mejor sus posibilidades dado que operaciones de load a una misma locación pueden reordenarse, con lo cual se podrá realizar el despacho de un load aun con operaciones de load previos cuyas direcciones no sean conocidas.

Además los requerimientos de la estructura de interconexión se verán significativamente reducidos, pues las complejas estructuras que antes mantenían la coherencia ya no son necesarias. Por otro lado el sistema de interconexión ya no necesita estar integrado a un controlador de coherencia, o conectado directamente al bus del sistema. Más aún, puede utilizarse una *System Area Network* (SAN) común, tecnología usualmente empleada en clusters, en lugar de emplear interconexiones de mayor complejidad.

En resumen, en el aspecto de la arquitectura, el uso de sistemas de memoria compartida no-coherentes redunda en sistemas más simples y menos especializados. Abrirá además la puerta al desarrollo de arquitecturas NUMA basadas en técnicas conocidas, provenientes de clustering, logrando una notoria reducción en los costos de desarrollo y de hardware, y ciertamente una mayor escalabilidad del sistema.

4. LíNEA DE TRABAJO

No son muchos los modelos de consistencia que no se soportan en la coherencia de cache. Básicamente se desarrollaron dos modelos, el modelo Dag [RDBR96], un modelo de consistencia relajado de aplicación en sistemas DSM para programación multithread, y el Location Consistency LC [GS00], en el que el estado de una locación de memoria se modela a través de un *partially ordered multiset* (pomset). Ambos modelos no cache coherentes, aunque desarrollados de manera independiente, resultan ser equivalentes según fue demostrado por Frigo [Fri97].

Tomados estos modelos como punto de partida,

nos enfocamos en los protocolos de manejo de cache. La consistencia Dag está basado en threads a nivel de usuario. Se implementó para Cilk, un lenguaje multithreaded basado en C, y adoptó un algoritmo de coherencia denominado Backer. Este algoritmo implementa tres operaciones básicas para manipular objetos compartidos: *fetch*, *reconcile* y *flush*. El primero copia un objeto en la cache y lo marca como *clean*. Un *reconcile* copia un objeto *dirty* de la cache a memoria y lo marca como *clean*. El *flush* desplaza un objeto *clean* de la cache. Es de notar que estas operaciones son locales, no involucran a las cache de los demás procesadores.

En realidad este algoritmo implementa un modelo de consistencia más fuerte que el Dag, y pensamos además que el proceso de invalidación total, al cual se vería sometida la cache asociada con un thread dependiente, afectará la performance del sistema. Además entendemos que una operación selectiva de copiar en memoria bloques de cache, del *reconcile*, demanda una política de write through en cache.

En lo que respecta al modelo LC, el protocolo de cache propuesto es tal que asegura que una operación de lectura devuelva un valor válido desde el *pomset*; bien sea el de la última escritura ordenada, o el correspondiente a cualquier escritura que no resulte ordenada respecto a la lectura. Su implementación se vale de tres estados para un bloque: *Invalid*, *Clean*, y *Dirty*, con acciones específicas para *acquire* y *release*. Un *release* pasa un bloque de *dirty* a *clean*, y un *acquire* fuerza un bloque *clean* a *invalid*. En este caso no sólo visualizamos problemas de implementación sino que, además, en coincidencia con Frigo [Fri97], entendemos al modelo no razonable al no confinar el no determinismo.

Estamos trabajando en un protocolo que siendo no cache coherente, resulte superador de las ineficiencias de los protocolos aludidos, y que sea soporte de un modelo de consistencia construible y razonable.

Referencias

[AF92] H. Attiya and F. Friedman. A correctness condition for high perfomance multiprocessors. *Proceedings of the Twenty-Forth ACM Symposium on Theory of Computing*, 1992.

[Fri97] Matteo Frigo. Magister of science tesis: The weakest reasonable memory model. *Massachusetts Institute of Technology*, 1997.

[GS97] Guang R. Gao and Vevek Sarkar. On the importance of an end-to-end view of memory consistency in future computer systems. *Proc. Int´l Symp. High Perfomance Computing*, 1997.

[GS00] Guang R. Gao and Vevek Sarkar. Location consistency-a new memory model and cache consistency protocol. *IEEE Transaction on Computers*, 2000.

[RDBR96] Christopher F. Joerg Charles E. Leiserson Robert D. Blumofe, Matteo Frigo and Keith H. Randall. Dag-consistent distributed shared memory. *Proceedings of the 10th International Parallel Processing Symposium*, 1996.