Caracterización del tráfico en protocolos de coherencia *snoopy*

M.J. Garzarán, V. Viñals, J.L. Briz, A. Orio
Centro Politécnico Superior - Univ. Zaragoza.
Mª de Luna, 3 50015 - Zaragoza
{garzaran,victor,briz}@posta.unizar.es

Resumen

En esta contribución exponemos la puesta a punto de un marco experimental para evaluar cuestiones relacionadas con jerarquía de memorias en multiprocesadores de memoria compartida. Para ello hemos examinado varios entornos de simulación, seleccionado una carga de trabajo, escrito modelos de memoria cache coherente para bus compartido y, finalmente, hemos confrontado los resultados con trabajos previos. Al margen de este objetivo de verificación, tambiénn estudiamos el tráfico generado por tres protocolos snoopy con un subsistema de memoria PRAM (multiples referencias con latencia unidad), para observar las diferencias entre invalidación y actualización, así como la utilidad del servicio de copias entre memorias cache.

Palabras clave: multiprocesadores simétricos, coherencia de caches, caracterización de protocolos de coherencia.

1. Introducción

Recientemente hemos iniciado una línea de investigación sobre jerarquía de memorias en multiprocesadores de memoria compartida (MMC), con la intención de extender nuestra experiencia en uniprocesadores con Memoria Cache (Mc) multinivel dentro del chip [Iba96, Jim96].

Para evaluar cualquier idea de diseño, es necesario un marco experimental. El primer objetivo de este trabajo es examinar varios entornos de simulación y seleccionar una carga de trabajo. A este respecto hemos revisado la literatura y la oferta de software de dominio público. Tras seleccionar MINT como simulador y SPLASH-2 como carga de trabajo, por las razones que se exponen mas adelante, hemos escrito unos modelos sencillos de memoria cache coherente, y hemos verificado nuestros resultados con trabajos previos.

Durante los últimos años se han desarrollado protocolos que aseguran la coherencia de datos entre las Mc y la Memoria Principal (Mp) compartida, basados en la colaboración de niveles del computador diferentes. Desde soluciones puramente hardware, hasta soluciones con poco hardware, pero que necesitan mucho trabajo de compilación y soporte de lenguaje máquina. Una buena panorámica del tema aparece en [Ste90]. Comercialmente predominan los protocolos hardware, distribuidos en las caches del sistema, y que reaccionan en base a la observación constante del Bus (*snoopy cache protocols*). Una escritura a una variable compartida puede ocasionar una invalidación o una actualización en las Mc que poseen esa misma variable; comercialmente la invalidación está mucho mas extendida.

Al margen del primer objetivo de verificación, hemos utilizado el método de caracterización propuesta por Woo et al. en [Woo95] para estudiar el tráfico generado por tres protocolos snoopy: uno basado en invalidación tipo Illinois [Pap84], otro basado en actualización tipo

Dragon [McC84] y un tercero basado en autoinvalidación tipo *competitive snoopy* [Kar88]. Este método consiste en considerar un subsistema de memoria ideal, el mismo que asume el modelo PRAM de cálculo, para observar las diferencias entre invalidación y actualización, así como la utilidad del servicio de copias entre memorias cache. Este estudio es pues relevante para comprender el compromiso invalidación/actualización en multiprocesadores basados en bus, también llamados multiprocesadores simétricos (MpS). Estás máquinas se caracterizan por una escalabilidad limitada a 16-32 procesadores, y por una importancia comercial creciente. Además las propuestas más ambiciosas de MMC pero físicamente distribuída (*DSM*), son interconexiones de nodos que a su vez son MpS.

La estructura de esta contribución es como sigue. En la Sec. 2 se exponen las técnicas de simulación más utilizadas, justificando la elección del entorno MINT mediante una pequeña comparativa con otros entornos públicos. En la Sec. 3 se comenta la elección de SPLASH-2 como carga de trabajo y se dan estadísticas resumen del subconjunto elegido. En la Sec.4 se repasa el método de caracterización utilizado. En la Sec. 5 se explican los tres protocolos utilizados junto con la notación empleada. En la Sec. 6 se aportan detalles sobre la verificación del marco de simulación (MINT + modelo jerarquía + SPLASH-2); en esta sección se comentan también las medidas de tráfico para cada protocolo de coherencia, descomponiendo el tráfico según su fuente (Mp, Mc) y su naturaleza (invalidación, actualización). La Sec. 7 concluye con una valoración del trabajo realizado.

2. Entornos de simulación

Existen dos técnicas principales para la simulación de MMC: simulación dirigida por traza o simulación dirigida por ejecución de programa. La primera técnica consiste en generar y almacenar una traza de referencias de un *benchmark* o Aplicación Paralela de Prueba (APP en lo que sigue). Después la traza alimenta a un simulador de la jerarquía de memoria que contabiliza los eventos de interés. Puesto que la traza se genera con una máquina real, su principal inconveniente es que no permite cambiar el entrelazado de las referencias según la respuesta funcional o temporal de la jerarquía, ni tampoco simular comportamientos de procesador diferentes del original (p.e. ejecución especulativa de `ld/st`). Además, las trazas ocupan mucho espacio, podemos hablar de decenas de GB.

La segunda técnica —simulación dirigida por programa— ejecuta en cada experimento el programa paralelo, a la vez que realiza la simulación. El entorno de simulación suele constar de un generador de referencias, un gestor de una cola de eventos, y el modelo del subsistema de memoria a simular. El generador ejecuta la APP y envía eventos al gestor, quien a su vez despierta al sistema simulado cuando se producen referencias a memoria, sincronizaciones, creación de procesos, etc. Cuando las operaciones asociadas a un evento terminan, el gestor devuelve el control al generador, de modo que algún proceso de la APP pueda continuar.

La generación de los eventos se consigue interpretando o contaminando el código. La contaminación consiste en insertar llamadas al gestor de eventos en el código de la APP. En general, la interpretación es más flexible y la contaminación más eficiente, porque el código contaminado se ejecuta de forma nativa, a la máxima velocidad posible.

Hemos examinado parte de la oferta pública de simuladores conducidos por programa, experimentando con MINT [Vee93, Vee94] y con Tango Lite [Her93]. En estos momentos estamos evaluando también AUGMINT [Sha96]. Hemos concluido que no era preciso desarrollar nuestro propio simulador, y hemos seleccionado MINT por las siguientes razones:

- El modelo de eventos y tareas que soporta MINT relaciona las actividades del simulador y de la APP de forma sencilla.
- Antes de final de este año se espera una revisión que amplíe el lenguaje máquina interpretado para MIPS-II y MIPS-III.
- Los simuladores desarrollados pueden usarse con AUGMINT, que soporta código i486.

2.1. MINT

MINT es un simulador dirigido por programa, pensado para experimentar con la jerarquía de memoria en MMC, que procede por interpretación. El algoritmo de interpretación identifica secuencias grandes de código sin `saltos` ni `ld/st`, y genera código nativo para el procesador sobre el que se ejecuta. En una nueva ejecución de esa secuencia, se desconecta la interpretación y se ejecuta el código nativo, aumentando la velocidad de simulación.

MINT utiliza un espacio único de direcciones, y simula múltiples procesos situando el espacio de datos, pila y código de cada proceso dentro de su propio espacio. Los ejecutables se obtienen a partir de programas objeto compilados para MIPS R3000 sobre IRIX, enlazados estáticamente, de forma que se incluye y simula el código de librerías. Como en todos los métodos que no utilizan microprogramación o monitorización hardware, se pierde el código de Sistema Operativo ejecutado en modo supervisor.

Para especificar una jerarquía de memoria en el entorno MINT, hay que considerar las siguientes abstracciones:

- *Procesos* : creados por la APP con semántica de `fork()`, en la que toda variable es privada a menos que se coloque en memoria compartida.
- *Tareas*: son las actividades sujetas a la planificación del gestor de tareas (p.e. una lectura). A cada tarea se le asocia una fecha de ejecución, una prioridad y una función.
- *Eventos*: el generador interpreta un proceso de la APP. Cuando se produce un evento de interés bloquea al proceso e instala una nueva tarea en la cola. El programador puede ligar a cada tarea la reacción del componente implicado asociando una función a la tarea. El valor de retorno de la función determina que el proceso continúe o se bloquee. De esta forma, el entrelazado de referencias está ligado a la temporización y comportamiento de la memoria, y la simulación tiene gran parecido con la realidad.

Es posible crear tareas, destruirlas, y generar eventos desde la propia APP. MINT permite utilizar en la APP tres primitivas de sincronización: cerrojos , barreras y semáforos. Para ello, intercepta las rutinas de librería de IRIX que sincronizan procesos creados con `sproc()`. Así, por ejemplo, si se implementa un cerrojo con la librería `test_and_set`, MINT no reconoce la sincronización y no activa la función del simulador asociada al evento de sincronización.

2.2. Otros simuladores

Tango Lite es un simulador dirigido por ejecución, desarrollado para MIPS R3000, que procede contaminando el fuente en ensamblador de la APP. Para simular el código de librería hay que disponer de los fuentes y contaminarlos, siendo por ello menos portable que MINT.

Para cada proceso que se crea en la APP existe un proceso Tango Lite, que distribuye su tiempo entre varias actividades: a) código de la APP, b) código añadido por contaminación y que genera referencias, c) gestor de eventos, y d) modelo de la jerarquía. Puesto que la planificación de tareas está distribuida entre los procesos Tango Lite, cada vez que se ejecuta una operación

interesante, como una lectura, el proceso Tango Lite debe realizar un cambio de contexto para ver si existe otro proceso más retrasado, y así conseguir un entrelazado realista de referencias.

En la literatura se citan algunas dificultades en el uso de Tango Lite [Ngu96]**:**

- La interfase para la escritura del modelo de la jerarquía no es sencillo. Los recursos de planificación de tareas no permiten modelar con facilidad varias referencias pendientes o modelos de consistencia relajados.
- No garantiza resultados repetitivos, ya que utiliza directamente las llamadas al sistema para la asignación dinámica de memoria y por tanto simulaciones sucesivas pueden tener, por ejemplo, conflictos diferentes en Mc.

Augmint es un simulador dirigido por la ejecución de la propia APP, pero la planificación de tareas y la interfase del simulador son idénticas a MINT (portabilidad de simuladores diseñados para MINT a Augmint). Se ha desarrollado para i486 bajo Linux, Solaris y Windows NT.

3. Carga de trabajo seleccionada

Lo más utilizado en entornos académicos para evaluar MMC es sin duda SPLASH-2, que consiste en un conjunto de programas de cáculo científico y gráfico, diseñados para ejecutar en MMC [Sin92, Woo95]. Las datos de entrada estándar en cada programa son un compromiso entre tamaño del problema realista y tiempo de simulación razonable. SPLASH-2 está escrito en C y utiliza, para la gestión de procesos y memoria compartida, las macros PARMACS del ANL [Lus87]. El grupo que mantiene SPLASH-2 proporciona una implementación de las macros para IRIX, que permite a MINT capturar los eventos de gestión de procesos y memoria compartida, según se ha explicado en la sección anterior. En cualquier caso, es posible experimentar con otras implementaciones de grupos cercanos solventes en este tema [Art97], p.e. para tener en cuenta la sobrecarga en tráfico de los cerrojos construidos a partir de instrucciones máquina de sincronización (`load-link` y `store-conditional` en el caso MIPS).

En la Tabla 2-1 se presenta el subconjunto de SPLASH-2 que hemos seleccionado, y varias medidas para 16 procesadores.

Aplicación	**Parámetros**	**Inst (M)**	**Lect. (M)**	**Escr. (M)**	**Barre-ras**	**Cerro-jos**
Radix	puntos 256 k radix =1024	14,05	3,35	1,92	10	182
Radiosity	room -ae 5000.0 -en 0.050 -bf 0.10	2746,28	574,72	306,30	10	212446
Ocean	rejilla 258x258, tolerance 10^{-7}'s, steps 4	282,30	81,60	18,53	364	1296
LU	matriz 512x512, bloques 16x16	340,63	97,63	47,82	66	0

Tabla 3-1: Subconjunto utilizado de SPLASH-2. Estadísticas para 16 proc. Radix y Radiosity son intensivos en cálculo entero; Ocean y LU son intensivos en coma flotante. LU es un núcleo de cáculo, no un programa completo. Compilación con `cc -O2 -mips2 -non_shared` (V. 7.1 de MIPS SGI).

4. Método de caracterización del tráfico

El espacio de diseño y evaluación de prestaciones en MMC es muy grande. Caracterizar programas implica conocer sus propiedades fundamentales y sus interacciones con la

arquitectura, con el objetivo de podar el mayor número posible de casos de experimentación, identificando situaciones irreales. En [Woo95] se proponen 4 ejes de caracterización, con la hipótesis de que son poco dependientes de la temporización de la jerarquía de memoria (ancho de banda, latencia y contención) : a) utilización de los procesadores, b) *working set (WS)*, c) relación comunicación/cálculo y d) localidad espacial.

A partir de la hipótesis de independencia temporal, se postula un modelo de simulación sencillo, en el que red, Mc y Mp responden a cualquier número de peticiones en un ciclo, y en el que el único freno a la concurrencia es la inherente del algoritmo (sincronización y cálculos replicados). Esta situación se corresponde al modelo PRAM de cálculo paralelo [For 78].

En la siguiente sección vamos a concentrarnos en la relación comunicación/cálculo, en forma de bytes por instrucción (o bien en forma de ancho de banda si asumimos un procesador de una cierta velocidad fija). Esta medida de tráfico, *sí* es dependiente del dimensionado de las Mc de cada procesador (nº de bloques, tamaño de bloque y asociatividad). En [Woo95] tras estudiar los *WS* de las aplicaciones se concluye: a) que en la mayoría de programas SPLASH-2, una Mc de 1MB puede contener sin trasiego la parte del problema que le corresponde (en 32 procesadores), y b) que si se escala a la vez tamaño de problema y nº de procesadores, esto sigue siendo cierto. Nosotros tomamos por tanto este punto de partida.

Por razones de espacio no presentaremos resultados para tamaños de Mc en los que el WS no cabe en la Mc, en el caso de Ocean y Radiosity (8KB según [Woo95]). Estos y otros datos, que permiten reconstruir nuestros experimentos y validar otros, se encuentran en http:// www.cps.unizar.es/~garzaran/inves/jjpar97

5. Protocolos simulados

Hemos llevado a cabo simulaciones de tres protocolos: invalidación MESI tipo Illinois [Pap84], actualización MOES tipo Dragon [McC84] y [Tha88], y actualización con autoinvalidación MOESI (*competitive snooping* [Kar88]) . Todos ellos aprovechan una línea de compartición que informa a la Mc que coloca un comando en el Bus sobre la existencia de otra(s) Mc que poseen (línea en estado sh) o no (línea en estado $\overline{\text{sh}}$) el bloque. Asimismo se asume un servicio de sumistro de bloques entre Mc, sin necesidad de la intervención de Mp.

Describiremos los protocolos mediante diagramas de estados, atendiendo a lo siguiente:

- Un acierto provoca una sola transición, y un fallo provoca dos transiciones: la 1ª hacia un estado de invalidación por reemplazo y la 2ª hacia el estado que corresponda.
- Un comando de Bus provoca una sola transición en caso de acierto
- Las transiciones tienen el siguiente formato, comentado en la Tabla 5-1:

evento: destino(comando, parámetros)

En cada estado, una aserción explica si el bloque está presente en una o varias Mc, y si es coherente o no con Mp. En la Tabla 5-2 se comenta la notación. Estas reglas se adaptan muy bien a la especificación de protocolos basados en directorio o de buses con transacciones no atómicas (*split transaction bus*) ya que la estructura de transiciones y estados es general y flexible[1].

1. Como soporte a la docencia ha demostrado ser eficaz, porque obliga a razonar en cada transición sobre tres elementos esenciales: 1) quién es el destinatario del comando, 2) qué se pide y 3) qué parámetros hacen falta. La notación clásica, basada en nombres de comando, es compacta, pero soporta peor razonamientos sobre correción. Las reglas presentadas se basan en el trabajo de C. Rodríguez [Rod92].

evento			destino	
rh	acierto en lectura		**Mp**	mem. principal
wh	acierto en escritura		**Mc**	mem. cache. Una, varias o todas, según el caso
rm	fallo en lectura		**all**	Mp y Mc´s
wm	fallo en escritura			
rpl	reemplazo			
comandos de Bus			**prámetro**	
rB	lect. de bloque		**x**	contenido del bloque referenciado
wB	escr. de bloque		**@x**	dirección del bloque referenciado
rBinv	lect. de bloque con invalidación	invali-dación	**x'**	palabra
inv	invalidación de bloque		**X = <x,@x>**	**X' = <x',@x'>**
rBw	lect. de bloque con escr. de palabra	actuali-zación	**u**	bloque víctima
wW	escr. de palabra			

Tabla 5-1: Notación para las transiciones que el procesador provoca en su propia memoria cache. Para las transiciones inducidas por el Bus, los eventos son los Comandos de Bus. Se llama **x** al bloque objeto de fallo o acierto y **u** al bloque víctima seleccionado para el reemplazo.

Estado	Aserción	Explicación
M	**1Mc≠Mp**	Copia única y diferente de Mp.
O	**nMc≠Mp**	Posibilidad de copias múltiples, iguales entre sí pero diferentes de Mp. La Mc es responsable de efectuar Copy-Back en caso de reemplazo.
E	**1Mc=Mp**	Copia única e igual en Mp.
S (invalidación)	**nMc=Mp**	Posibilidad de copias múltiples, iguales entre sí y con Mp.
S (actualización)	**nMc?Mp**	Posibilidad de copias múltiples, iguales entre sí pero diferentes *o no* de Mp. La responsabilidad de Copy-Back se delega en el bloque -único- en estado **O.**
	n ≥ 1	

Tabla 5-2: Explicación de los estados MOESI en cuanto al nº de copias de un bloque, su coherencia con Mp y la responsabilidad en caso de reemplazo.

Invalidación MESI tipo Illinois. Ver **Figura 5-1.** Para contar el tráfico, hemos supuesto que los comandos `rB` y `wB` generan un tráfico de 70B en el bus (6B de dirección + 64B de tamaño de bloque), mientras que `inv` genera un tráfico de 6B (dirección + comando).

Actualización MOES tipo Dragon. Ver **Figura 5-2**. El comando `wW`, utilizado para actualizar palabra entre caches genera un tráfico de 14B: (6B dirección + 8B palabra). El comando `rBwW` aparece en fallos en escritura, que requieren la lectura de bloque no presente y la posterior modificación de palabra: 84B (70B como en `rB` + 14B como en `wW`).

Actualización con autoinvalidación (*competitive snooping*) MOESI. Este tipo de protocolos van actualizando un bloque hasta que el tráfico de actualización acumulado se iguala al tráfico necesario para resolver un fallo: en ese momento, el protocolo "autoinvalida" al bloque. Una implementación de esta idea consiste en asociar un contador a cada bloque, inicializado con un valor que disminuye cada vez que dicho bloque es actualizado. Si el procesador referencia al bloque, el contador vuelve al valor inicial. Cuando el contador se hace cero, el bloque se autoinvalida.

Existen dos variaciones clásicas [Egg89], según se decremente sólo el contador de una Mc de

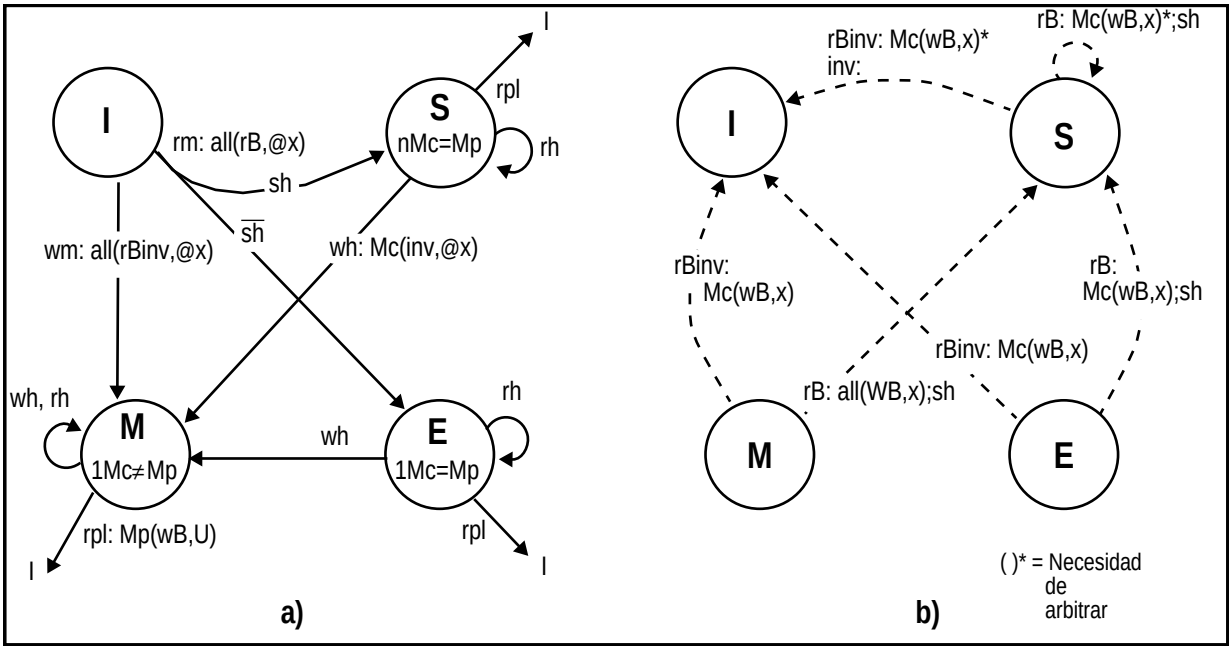


Figura 5-1: Protocolo MESI de invalidación con línea de compartición (sh) y suministro de bloques entre Mc. (a) Actividad en Mc debida al procesador. (b) Actividad en Mc inducida por los comandos de Bus.

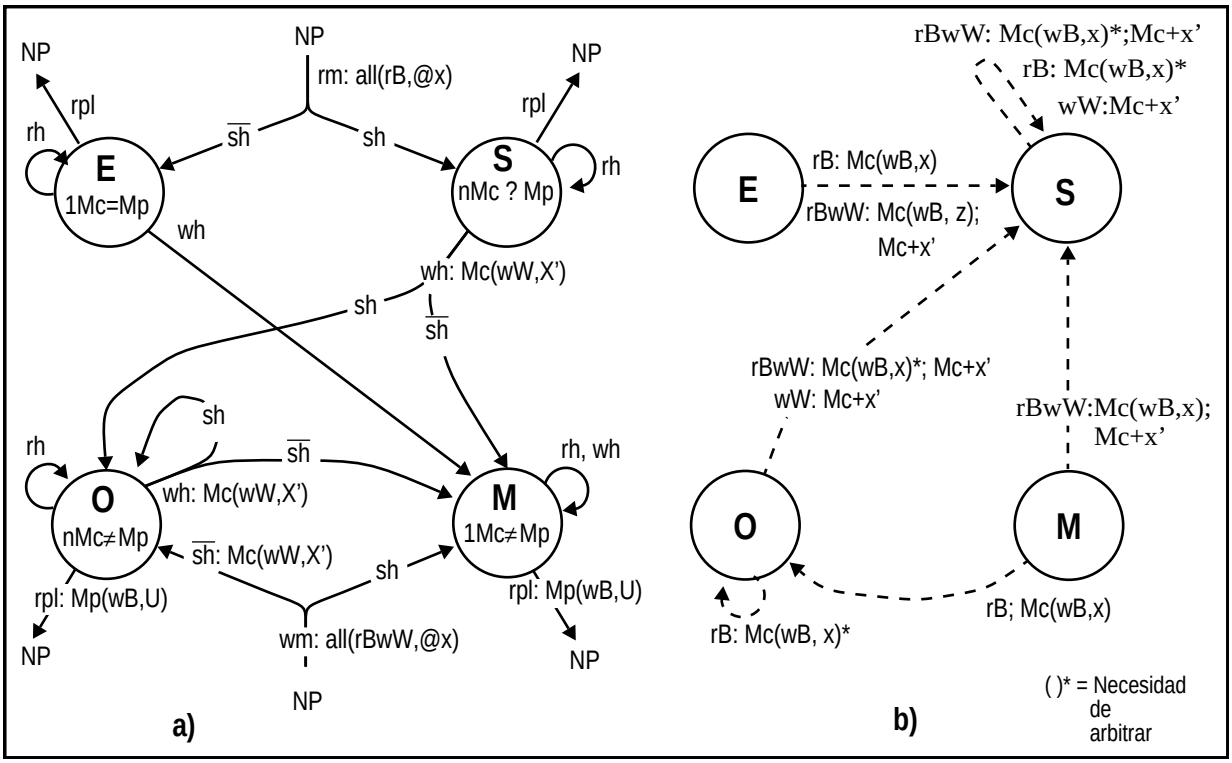


Figura 5-2: Protocolo MOES de actualización con línea de compartición (sh) y suministro de bloques entre Mc. (a) Actividad en Mc debida al procesador. (b) Actividad en Mc inducida por los comandos. En *todas* las transiciones inducidas la Mcactiva la línea sh; en el dibujo no se explicita.

las que tienen la copia, elegida arbitrariamente *(standard snoopy caching*), o bien se decrementen todos los contadores (*snoopy reading*), que es la aproximación seguida en [Cul97] y también en este trabajo. A fin de poder comparar con [Cul97] inicializamos el contador a 4.

6. Resultados: caracterización del tráfico

En la **Tabla 6-1** se muestran las transiciones entre estados por cada 100 referencias a memoria para el protocolo de invalidación, así como la tasa de fallos y la utilización de los procesadores. Los datos obtenidos son coherentes con los resultados que aparecen en [Cul97] que evalúan este protocolo con parámetros similares utilizando Tango Lite, si bien en esos trabajos no se especifica el código objeto generado (MIPS-I o MIPS-II)[1].

Aplicación	De/a	NP	I	E	S	M	Tasa Fallos (%)	Utiliz. proces.
RADIX	NP	0	0	0,00076	0,08604	0,44161	0,609	0,875
	I	0,00002	0	0	0,07973	0,00173		
	E	0	0	0,01572	0,00053	0		
	S	0	0,09948	0	8,5965	0,06803		
	M	0	0,00267	0	0,09704	90,70989		
LU	NP	0	0	0,00004	0,06783	0,00008	0,069	0,770
	I	0,00349	0	0	0,00106	0,00003		
	E	0	0	0,06213	0,00007	0,0127		
	S	0,00489	0,00981	0	31,46736	0,0098		
	M	0	0,00003	0	0,02247	68,37897		
OCEAN	NP	0	0	0,12267	0,10255	0,16737	0,572	0,991
	I	0,06712	0	0	0,17925	0,00016		
	E	0,01907	0	1,4309	0,00255	0,09866		
	S	0,04531	0,24585	0	14,17716	0,22137		
	M	0,26094	0,00016	0	0,22763	83,4999		
RADIOSITY	NP	0	0	0,00134	0,02121	0,00233	0,080	0,992
	I	0,00173	0	0	0,05304	0,00249		
	E	0	0,00006	0,00288	0,00001	0,00122		
	S	0,00085	0,06475	0	18,96737	0,02588		
	M	0,00189	0,00231	0	0,02897	80,92225		

Tabla 6-1: Transiciones entre estados por cada 100 referencias en el protocolo de invalidación MESI, 16 procesadores con caches de 1MB, asociatividad 4, tamaño de bloque 64 B y reemplazo LRU.

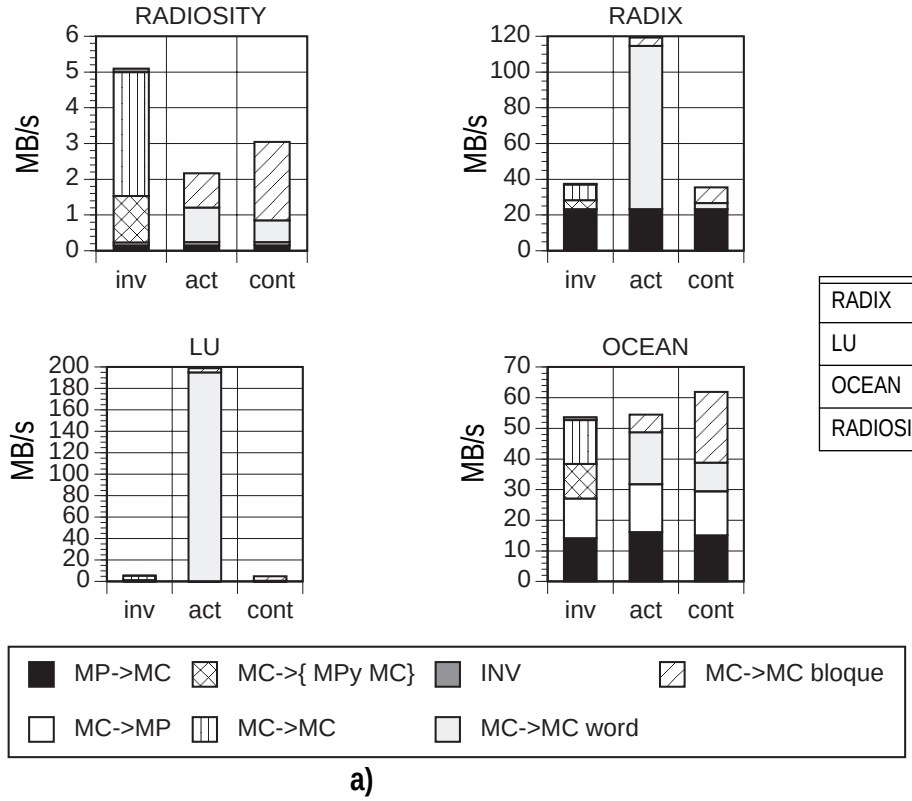
En la Figura 6-1 se muestra el ancho de banda calculado a partir de los pesos de cada transición y suponiendo que los procesadores ejecutan cualquier instrucción en un ciclo de 5ns (200 MIPS). Este tráfico debe ser escalado por la utilización de los procesadores para obtener una cota superior del tráfico que aparecería en una jerarquía con temporización real.

Para invalidación, descomponemos el tráfico en: a) carga de bloques no residentes en ninguna Mc (`Mp->Mc`), b) reemplazos (`Mc->Mp`), c) suministro de bloque entre Mc (`Mc->Mc`), d) suministro de bloque desde Mc hacia Mc y Mp (`Mc->Mp y Mc`), e) sobrecarga de invalidación (`inv`). En la tabla adjunta en la Figura, el tráfico (`Mc->Mc`) a su vez, está dividido en tráfico debido a `rB` y a `rBinv`. Además, se presenta el tráfico debido a Recarga por Bloques Invalidados (`RBI`), es decir, aquellos bloques que han sido previamente invalidados en una Mc y que cuando el procesador los vuelve a referenciar, todavía queda su huella porque no han sido elegidos como víctimas.

Para los dos protocolos de actualización, descomponemos el tráfico en: a) carga de bloques

1. Otra dificultad para validar simulaciones ha sido el punto de inicio de medidas en Radix, ya que en el indicado en la aplicación (tras el primer `barrier`) la discrepancia en nº de instrucciones es grande, y en otro punto razonable (creación de procesos), el tráfico se parece más, pero el nº de instrucciones difiere. Hemos escogido el primer punto, aunque no siempre los resultados son coherentes con [Cul97, Woo95].

no residentes en ninguna Mc (`Mp->Mc`), b) reemplazos (`Mc->Mp`), c) suministro de bloque entre Mc (`Mc->Mc bloque`) , y d) sobrecarga de actualización (`Mc->Mc word`).



	MC->MC		RBI
	rBinv	rB	
RADIX	0,14	8,71	4,28
LU	0,00	4,11	0,06
OCEAN	0,75	13,50	8,90
RADIOSITY	0,12	3,33	2,49

b)

Figura 6-1: a) Tráfico para los protocolos, inv: invalidación; act: actualización; cont: actualización-invalidación. b) Descomposición del tráfico Mc->Mc según el comando que lo provoca en invalidación; RBI (Recarga de Bloques Invalidados) indica el tráfico generado por referencia a bloques previamente invalidados.

El tráfico (`Mp->Mc`) se mantiene constante en los tres protocolos. Por otra parte, se ve útil el servicio de copia entre caches (`Mc->Mc`) en invalidación. El caso menos favorable para invalidación se presenta en Radiosity, donde el tráfico debido a RBI es proporcionalmente el más alto. El tráfico por invalidación (`inv`) es sin embargo muy bajo en todos los casos.

La mayor diferencia entre los tres protocolos se presenta en LU; parece que las caches guardan durante toda la ejecución bloques poco utilizados, pero que son escritos por otros procesadores. El protocolo de actualización-invalidación se muestra útil con respecto a actualización en los casos LU y Radix (en Radix incluso disminuye el tráfico con respecto a invalidación), en cambio, para Ocean y Radiosity este protocolo invalida demasiado pronto bloques que luego son referenciados, lo que provoca un aumento del tráfico de bloques entre Mc.

Ocean es la única aplicación que presenta tráfico notable de reemplazo, observación coherente con el porcentaje significativo de los fallos de capacidad en los estudios de referencia.

7. Conclusiones

Como resultado del trabajo expuesto estamos en posición de salida para extender nuestros modelos coherentes de gestión de dos niveles de Mc on-chip a un entorno multiprocesador. Con la experiencia adquirida suponemos que podríamos explotar cualquier otro simulador con un

esfuerzo inferior en un orden de magnitud al realizado. Metodológicamente parece correcto iniciarse en el tema con simulaciones independientes de la temporización, primero porque son menos complejas, y segundo, porque se captan propiedades de los algoritmos paralelos independientes de la arquitectura. La notación expuesta en la Sección 5 nos ha resultado mas útil que los clásicos comandos compactos, ya que al detallar las propiedades importantes de cada transacción, permite un buen nivel de abstracción, pero a la vez facilita la implementación. Respecto a la parte experimental, ha cumplido su objetivo de validar la corrección de simuladores y modelos de coherencia, a través de la comparación con los trabajos de referencia realizados en Stanford. Aunque la descomposición en tráfico que realizamos es original, los escasos puntos de diseño examinados no permiten extrapolar conclusiones generalizables.

Referencias

[Art97] E. Artiaga, N. Navarro, X. Martorell, Y. Becerra. Implementing PARMACS Macros for Shared Memory Multiprocessor Environments. Report UPC-DAC 1997-7.

[Cul97] D. Culler, J.P. Singh y A. Gupta. Alpha draft on Parallel Architecture. Dirección electrónica: `http://http.cs.Berkeley.edu/~culler/`

[Egg89] S. Eggers y R. Katz. A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation. En Proc. of the 15th Ann. Int. Symp. on Computer Architecture, Mayo 1988:373-382.

[For78] S. Fortune y J. Wyllie. Parallelism in Random Access Machines. En Proc. of the 10th Ann. ACM Symp. on Theory of Computing, 1978:114-118

[Her93] S. A. Herrod. Tango Lite: A Multiprocessor Simulation Environment. Introduction and User´s Guide. Technical Report, Stanford Universiy, Computer Systems Laboratory, Nov. 1993.

[Iba96] P. Ibáñez y V. Viñals. Performance Assessment of Contents Management in Multilevel On-Chip Caches. En Proc. of the 22nd EUROMICRO Conference. Praga, 2-5 Sept., 1996 : 431-440

[Jim96] L. M. Jimeno, P. Ibáñez y V. Viñals. Warm Time Sampling: Fast and Accurate Cycle-Level Simulation of Cache Memory. In Proc of the 22nd EUROMICRO Conference. Short Contributions. Praga, 2-5 Sept., 1996 : 39-44

[Kar88] A.R. Karlin, M.S. Manasse, L. Rudolph y D.D. Sleator. Competitive Snoopy Caching. Algoritmica 3, 1988:79-119.

[Lus87] E.L.Lusk y R. A. Overbeek, . Use of Monitors in FORTRAN: A Tutorial on the Barriers, Self-scheduling DO-Loop, and Askfor Monitors. Technical Report Num. ANL-84-51,Rev. 1, Argonne National Laboratory, Junio 1987.

[McC84] E. McCreight. The Dragon Computer System:An Early Overview. Technical Report, Xerox Corporation, Sept. 1984.

[Ngu96] A.T. Nguyen, M. Michael, A. Sharma y J. Torrellas. The Augmint Multiprocessor Simulation Toolkit for Intel x86 dirección electrónica: `ftp.cs.uiuc.edu:pub/research-groups/csrd/iacoma`.

[Pap84] M. Papamarcos y J. Patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In Proc.s of the 11th Int. Symp. on Computer Architecture, 1984: 348-354.

[Rod92] C. Rodríguez y V. Viñals, Curso de Doctorado “Jerarquías de memoria para uni y multiprocesadores“, Ciclo de Seminarios del Dpto de Arq. de Comp., DAC- UPC, Barcelona, Mayo 1992.

[Ste90] P. Stenström. A Survey of Cache Coherence Schemes for Multiprocessors. IEEE Computer, Vol. 23, No. 6, Junio 1990:12-24.

[Sin92] J. Singh, W.D. Weber y A. Gupta.. SPLASH: Stanford Parallel Applications for Shared-Memory. Computer Architecture News. Marzo 1992:5-44.

[Sha96] A. Sharma. Augmint, A multiprocessor simulator. Thesis. M.S., University of Illinois at Urbana Champaign, 1996

[Tha88] C. Thacker, L. Stewart y E. Satterthwaite. Firefly: A Multiprocessor Workstation. IEEE Transactions on Computers, vol. 37, no. 8, Agosto 1988: 909-920.

[Vee93] J. Veenstra y R. Fowler. MINT Tutorial and User Manual. Tech. Report 452, University of Rochester, Junio1993.

[Vee94] J. Veenstra y R. Fowler. MINT : A front end for efficient simulation of shared-memory multiprocessors. En Proc. of the 2nd Int. Workshop on Modeling, Analysis, and Simulation of Computer and Telecomm. Systems 1994:201-207

[Woo95] S. Woo, M. Ohara, E. Torrie, J. Singh, y A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proc. of the 22nd Int. Symp. on Computer Architecture, 1995 : 24-36.