Evaluación de un servidor web multiprocesador mediante Simics

Francisco J. Villa, Manuel E. Acacio, José M. García

***Resumen*–Hoy en día el uso de sistemas multiprocesadores no se limita a las tradicionales aplicaciones de ámbito científico para las que venían siendo usados, sino que dicho ámbito se ha visto ampliado con el creciente uso para la ejecución de aplicaciones comerciales. Se hace pues importante el poder estudiar el comportamiento de estas aplicaciones bajo determinadas cargas de trabajo, para lo que necesitamos disponer de un simulador que nos permita trabajar con todos los conceptos involucrados en la ejecución de las aplicaciones comerciales: CPU, jerarquía de memoria, sistema operativo, sistema de E/S y red de interconexión. En este trabajo presentamos nuestras primeras experiencias utilizando *Simics*, un simulador que permite simular sistemas multiprocesadores completos abarcando todos los aspectos antes mencionados, y mostramos un estudio sobre el comportamiento de un servidor de contenidos web estáticos. Los resultados obtenidos muestran el aumento de rendimiento que se obtiene usando un servidor biprocesador frente a un monoprocesador, a la vez que nos permiten comprobar las limitaciones que sufre *Simics*.**

***Palabras clave*–*Simics*, aplicaciones comerciales, simuladores de sistemas completos**

I. Introducción

Los sistemas multiprocesador han sido usados tradicionalmente en campos de aplicación eminentemente científicos: estudio y modelado del clima, modelado del universo, algoritmos moleculares, etc. Este tipo de problemas pueden ser reproducidos y estudiados muy fácilmente mediante el uso de simuladores como RSIM [8] y bechmarks como los contenidos en la suite SPLASH-2 [14].

Sin embargo, hoy en día los sistemas multiprocesador también están siendo usados para ejecutar otro tipo de aplicaciones, que podemos denominar comerciales, entre las que se encuentran por ejemplo los servidores web. Bien es sabido que en los últimos años la importancia de la red Internet ha ido creciendo casi exponencialmente, hasta el punto de convertirse en un integrante más de nuestras vidas. Actualmente, todas las medianas y grandes empresas, e incluso las pequeñas, disponen de un portal web que sirve de escaparate hacia sus clientes. Esta situación es extensible a todo tipo de organizaciones: gubernamentales, instituciones académicas, etc. Las grandes organizaciones, que esperan recibir un gran número de conexiones de usuarios diariamente, necesitan disponer de un equipo servidor potente, que en ocasiones es un multiprocesador. Otras aplicaciones para las que el uso de arquitecturas multiprocesador resulta esencial hoy en día son las aplicaciones de procesamiento de transacciones en línea (OLTP), donde se hace uso de bases de datos.

Debido al crecimiento que está experimentando este nuevo uso, se hace importante el poder modelar las aplicaciones comerciales de forma sintética. Al contrario que las aplicaciones de tipo científico, las comerciales presentan una serie de dificultades para su estudio: la actividad del sistema operativo es muy importante, así como la interacción con la jerarquía de memoria, sistema de almacenamiento y red de comunicación. Por tanto, si queremos estudiar una aplicación comercial necesitamos usar un simulador que nos permita modelar un sistema completo: sistema operativo, jerarquía de memoria, almacenamiento, buses, microprocesador, red de comunicación, etc. Esto es posible mediante la utilización de *Simics* [9], un simulador de sistemas completos que permite simular todos los aspectos mencionados anteriormente, y que es usado en la actualidad por más de 300 universidades en todo el mundo.

En este artículo estudiamos las posibilidades que ofrece *Simics* para la caracterización de este tipo de aplicaciones, describiendo en primer lugar las principales características del simulador. Tras familiarizarnos con la herramienta evaluaremos el rendimiento de un servidor web biprocesador, comparándolo con un monoprocesador.

El resto del artículo está organizado como sigue. En la siguiente sección comentaremos una serie de antecedentes relacionados con la evaluación de aplicaciones comerciales mediante *Simics*. En la sección III presentaremos las características más importantes del simulador. En la sección IV describiremos la carga comercial usada: un servidor de contenido web estático, siendo el servidor web usado *Apache* y la herramienta que sitúa la carga de trabajo en el mismo *httperf*. La sección V contiene los resultados de las pruebas realizadas. Finalmente, presentaremos en la sección VI las conclusiones de nuestro trabajo.

II. Antecedentes

Hasta hace poco tiempo la metodología usada para evaluar cargas comerciales consistía en generar trazas de las aplicaciones, usando dichas trazas para alimentar un simulador de nivel de usuario. Por ejemplo, en [12] se realiza un estudio del rendimiento de sistemas OLTP y de ayuda a la decisión usando esta metodología.

Con la aparición de simuladores de sistemas completos como *Simics*, la evaluación de cargas comerciales se simplifica, como podemos ver en algunos trabajos publicados recientemente. En [2] los autores realizan un

Francisco J. Villa es becario del Departamento de Ingeniería y Tecnología de Computadores de la Universidad de Murcia. Manuel E. Acacio y José M. García son profesores de dicho departamento. E-mail: {fjvilla, meacacio, jmgarcia}@ditec.um.es.

estudio del benchmark TPC-C utilizando este simulador, a la vez que describen uno de los problemas propios de la simulación de aplicaciones comerciales: la variabilidad exhibida por las mismas. Un análisis más profundo de dicho problema aparece en [4], donde se argumenta que la variabilidad viene dada en gran medida por las siguientes razones:

- El sistema operativo puede tomar diferentes decisiones de planificación.
- Los cerrojos pueden ser adquiridos en distinto orden, lo cual puede llevar a que en una ejecución se produzca una contención significativa por un recurso y en otra no.
- Puede ser que en una ejecución una transacción finalice durante un intervalo de medida, pero en otra no.

Alameldeen y otros [3] estudian el comportamiento de cinco aplicaciones comerciales diferentes, entre las que se encuentra un servidor web con contenidos estáticos, incidiendo de nuevo en el indeterminismo de las cargas comerciales.

III. El simulador Simics

Simics es una plataforma que permite el desarrollo tanto de hardware como de software, proporcionando los elementos necesarios para la simulación de ambos componentes dentro de un mismo contexto.

Esta herramienta permite simular varias arquitecturas (tanto monoprocesador como SMP), así como ejecutar sobre ellas sistemas operativos y aplicaciones comerciales, que pueden variar desde benchmarks tales como SPEC CPU2000 o TPC-C, hasta aplicaciones de escritorio o juegos. Es esta precisamente una de la características más interesantes del simulador, ya que podemos usar cargas de trabajo comerciales, algo que no es posible en otros simuladores como RSIM [8].

A. Arquitecturas simuladas

Simics simula en total nueve arquitecturas de procesadores: UltraSparc II, UltraSparc III, x86, AMD x86-64, Alpha, PowerPC, IA-64, ARM y MIPS. Usando la arquitectura UltraSparc III podemos alcanzar un límite teórico de 384 procesadores. Las pruebas realizadas en este trabajo se basan en la arquitectura x86, en la que el límite teórico está en 15 procesadores.

B. Modos de ejecución e interfaces de modelado del tiempo

Simics es un simulador dirigido por eventos, con una resolución de tiempo de un ciclo de reloj, siendo la longitud de dicho ciclo definible por el usuario. Un evento es una interrupción de un dispositivo o la ejecución de una instrucción, por ejemplo. Podemos planificar que suceda un evento después de que se hayan ejecutado cierto número de pasos (o steps). Un paso o step es una instrucción que se completa, una instrucción que genera un excepción o una interrupción externa.

Una definición importante es la de tiempo de ejecución de una instrucción. En el caso de un *monoprocesador* este tiempo viene determinado por el *modo de ejecución* y las *interfaces de medición del tiempo* que estemos usando. El caso de los *multiprocesadores* es más complicado porque cada procesador puede tener su propia noción del tiempo. En *Simics*, se ha optado por serializar la ejecución en multiprocesadores por cuestiones de rendimiento. Lo que se hace es asignar un quantum de ejecución a cada procesador, pasado el cual la ejecución se traslada al siguiente procesador. Este quantum viene expresado en ciclos y es configurable por el usuario. Como consecuencia de la serialización que realiza *Simics*, se mantendrá la consistencia secuencial [7] como modelo de consistencia de memoria.

El simulador proporciona dos modos de ejecución: ejecución en orden, que está disponible para todos los procesadores, y ejecución fuera de orden, que está disponible sólo en los procesadores SPARC. En el modo de ejecución en orden cada instrucción se convierte en un único evento y las instrucciones son planificadas siguiendo el orden del programa. Por tanto, si una instrucción se detiene un número determinado de ciclos (por ejemplo por un fallo de caché), las instrucciones que hay a continuación deberán esperar a que se complete la ejecución de la instrucción detenida. Una instrucción puede detenerse en la fase de búsqueda o en la de acceso a datos. Si estamos simulando un multiprocesador, es posible que una instrucción esté parada cuando se acaba el quantum asignado al procesador. En este caso es posible que otro procesador pueda manipular la instrucción que está por completar.

En el modo de ejecución fuera de orden la instrucción es dividida en varias fases, que pueden pertenecer a estas categorías: el evento de inicio, el evento de finalización, y cualquier otra fase entre estos dos eventos. El evento de inicio toma el valor del PC y busca la instrucción correspondiente. El evento de fin ocurre cuando todas las instrucciones previas han concluido y la ejecución de la instrucción ha terminado. El resto de eventos no son visibles por el usuario y se completan tan pronto como las dependencias han sido resueltas.

Estrechamente ligadas a los modos de ejecución tenemos las interfaces de modelado del tiempo, mediante las cuales podemos especificar cómo se determinan las latencias de los eventos antes descritos. Por defecto, *Simics* proporciona dos abstracciones genéricas que relacionan el tiempo simulado con el número de instrucciones ejecutadas. Cuando usamos el modo de ejecución en orden cada instrucción toma exactamente un ciclo de reloj por defecto. En el caso de un multiprocesador esto significa que en un momento determinado todos los procesadores habrán ejecutado el mismo número de instrucciones. En el modo de ejecución fuera de orden no hay correspondencia definida entre el número de ciclos transcurridos y el número de instrucciones ejecutadas. En este modo de ejecución se usa un controlador de consistencia para observar las dependencias de memoria. Este controlador fuerza un modelo al menos tan conservador como el *Total Store Ordering* (TSO1) [7].

Por defecto no hay modelos de medición del tiempo del sistema de memoria, lo cual motiva que los accesos a memoria sean detenidos cero ciclos, y cada instrucción consuma un ciclo. El simulador ofrece la posibilidad de crear una jerarquía de memoria, gracias a la cual esta limitación puede ser salvada.

C. Creación de una jerarquía de memoria

Las funcionalidades que ofrece *Simics* están agrupadas en *módulos*. Un módulo es un fichero que está escrito en C y que implementa una clase que define un tipo de objeto. Existe un módulo que implementa la clase *generic-cache*, el cual define un tipo de objeto caché. Cada objeto de la clase *generic-cache* tiene una serie de atributos que nos permiten configurarlo como deseemos. Haciendo uso del atributo *next-level* podemos crear jerarquías de caché, para lo cual dicho atributo debe apuntar al siguiente objeto en la jerarquía de memoria. El número de niveles en la jerarquía de cachés no está limitado, y la única limitación es que no se recomienda que dos procesadores compartan una misma caché.

Otros atributos nos permiten controlar el número de líneas de la caché, el tamaño de la línea, la asociatividad, la política de escritura (write-back o write-through), los tiempos de acierto y las penalizaciones por fallo. Asimismo, debemos indicar a qué procesador se conecta la caché. En el caso de que estemos modelando un multiprocesador SMP el protocolo de coherencia usado es MESI.

D. Simulación de una red

Simics proporciona la posibilidad de simular varios nodos interconectados mediante una red de área local. Para ello disponemos de un módulo denominado *ethernet-central*, al cual podemos considerar como la red de interconexión (una red simulada). El proceso es sencillo: cuando ejecutamos el módulo *ethernet-central*, éste queda a la espera de que se produzcan peticiones de conexión. Cuando simulemos un nodo, simplemente tendremos que conectarlo a la red mediante el comando correspondiente.

E. Instalación y ejecución de software en un sistema simulado

En *Simics*, la instalación y ejecución de software se realiza de la misma forma que en un entorno real. Una vez que hemos instalado el software en un sistema simulado tendremos que guardar los cambios introducidos en el fichero imagen para que estén disponibles en nuevas simulaciones. Existen dos formas de realizar esto: crear un *checkpoint* de la simulación o guardar los cambios en un *fichero diferencial* de la imagen. Mediante un checkpoint podremos recuperar la simulación justo en el punto en que se creó el mismo. El principal inconveniente de esta opción es que no podremos variar la configuración del sistema simulado, ya que hacer esto una vez que la simulación ha comenzado puede producir resultados imprevistos. Mediante la creación del fichero diferencial, dispondremos de un fichero que podremos añadir a la imagen de disco original cuando comencemos la simulación.

IV. Entorno de trabajo

A. El servidor web Apache

Apache [5] es un servidor de contenidos web estáticos y dinámicos, si bien en las pruebas presentadas en este trabajo se ha utilizado solamente como servidor de contenidos estáticos. El servidor se ha compilado incluyendo todas las opciones indicadas por el grupo de desarrollo del mismo para aumentar el rendimiento [6]. Entre estas opciones está la utilización del módulo de multiprocesamiento *worker*, con el que las peticiones son atendidas mediante hilos en vez de procesos (que es la opción por defecto). También se ha configurado el servidor para mantener un pool de hilos inactivos de forma que cuando llegue una petición pueda ser atendida sin tener que esperar a que se cree un nuevo hilo.

B. Benchmarks usados

La utilidad *httperf* [10] es una herramienta para medir el rendimiento de un servidor web. En su modo de funcionamiento básico, *httperf* genera un número fijo de peticiones HTTP de tipo GET y mide cuántas respuestas y a qué velocidad llegan desde el servidor. Las opciones más importantes que nos permite este programa son:

- Permite simular la existencia de usuarios distintos, o lo que es lo mismo, el concepto de sesión. Dentro de cada sesión podemos especificar el número de peticiones que se van a realizar, así como la velocidad a la que se emitirán.
- Permite especificar la tasa fija a la que se van a crear las conexiones o sesiones.
- Podemos hacer que las peticiones se realicen a *N* páginas distintas.

Como medida del rendimiento usada, el programa indica el *tiempo medio de respuesta*, definido como el tiempo medio entre el envío del primer byte de la petición y la recepción del primer byte de la respuesta.

C. Metodología

Las pruebas realizadas consistirán en variar la configuración del servidor gracias a *Simics*, y usando *httperf*, ejecutar la misma prueba para cada configuración. Esta metodología presenta un inconveniente: la forma en que *httperf* genera las peticiones es aleatoria, lo cual provoca que sucesivas ejecuciones de esta herramienta darán como resultado diferentes trazas de peticiones. Esto nos lleva a comparar las arquitecturas simuladas usando tests que no son completamente iguales. La forma de asegurar que en cada prueba se realizan las mismas peticiones y en el mismo orden es generar un fichero con la traza de las peticiones realizadas. Realizando una prueba base capturamos las peticiones generadas y las escribimos en un fichero; en pruebas posteriores podremos usar este archivo para recuperar las peticiones en el mismo orden en que fueron generadas en la prueba base.

Para manejar el problema de la variabilidad comentado en la sección II, se aplicarán técnicas estadísticas estándar [4]. Cada prueba se repite cinco veces, tomándose como resultado de una prueba la media aritmética de las distintas simulaciones realizadas. Además, es necesario usar la desviación típica como una medida que nos permita descartar aquellas pruebas que se alejen demasiado de la media.

D. Configuración hardware

El estudio se llevará a cabo para tres arquitecturas servidor distintas: dos arquitecturas monoprocesador en

las que el tamaño de la caché L2 será de 512 KB y 1024 KB respectivamente y una arquitectura biprocesador en la que cada procesador cuenta con una caché L2 de 512 KB. En la tabla I podemos ver los parámetros básicos del sistema.

TABLA I

PARÁMETROS BÁSICOS DE LOS SERVIDORES SIMULADOS

Microprocesador	
Frecuencia de reloj	500 Mhz
Modelo	Pentium 3
Caché L1	
Tamaño línea	64 bytes
Tamaño caché	32 KB
Política de escritura	WB
Asociatividad	Mapeo directo
Tiempo de acierto	2 ciclos
Penalización por fallo	10 ciclos
Caché L2	
Tamaño línea	64 bytes
Tamaño caché	512 KB/1024 KB
Política de escritura	WB
Asociatividad	4
Tiempo de acierto	10 ciclos
Penalización por fallo	50 ciclos
Memoria principal	
Tamaño	512 MB
Sistema operativo	
Sistema operativo	Red Hat 7.3
Versión núcleo	2.4.18-3
Red	
Modelo	Ethernet 10 Mb/s

Tanto el servidor como el cliente serán máquinas virtuales simuladas por *Simics*. La interconexión entre dichas máquinas se realizará mediante un red Ethernet de 10 Mb/s que también será simulada.

V. ESTUDIO DEL RENDIMIENTO

A. Curva de respuesta de Apache en función del número de peticiones recibidas

El propósito de esta sección es estudiar cómo varía el tiempo de respuesta por petición de Apache en función del número de peticiones simultáneas que se realicen al servidor, para cada una de las arquitecturas propuestas en la sección anterior. Este estudio se podría haber realizado mediante máquinas reales; sin embargo, utilizando *Simics* conseguimos mayor flexibilidad para variar la configuración del sistema analizado. Además, es posible obtener estadísticas sobre las CPUs y cachés, algo que no se puede hacer en un entorno real y que nos permite analizar más en profundidad los resultados obtenidos.

En total se realizarán 1000 peticiones sobre 10 páginas web cuyos tamaños varían entre 33'8 KB y 545 KB, siendo el tamaño medio de las páginas de 110 KB. Se ha generado un fichero de trazas de peticiones en el que las mismas se realizan siguiendo un patrón aleatorio. Además, debemos tener en cuenta el uso de *timeouts* para las peticiones: si transcurrido cierto tiempo el servidor no ha respondido, entonces podemos dar por muerta la conexión y cerrarla, incrementando el número de errores por timeout. El programa *httperf* permite que se especifique un valor de timeout, realizando automáticamente un control del mismo. En general, se recomienda un valor de timeout entre 5 y 15 segundos [11], siendo el valor usado en estas pruebas de 10.

Se han realizado ocho pruebas para cada arquitectura servidor, en las que el número de peticiones por segundo lanzadas sobre *Apache* ha sido 25, 50, 75, 100, 125, 150, 175 y 200 respectivamente. En la figura 1 podemos ver la evolución del número de peticiones atendidas en función del número de peticiones lanzadas para cada prueba y arquitectura. Dicha métrica es proporcionada por el propio servidor *Apache*.

De la figura 1 podemos extraer que el biprocesador es capaz de procesar entre un 7% y un 30% de peticiones más que el servidor monoprocesador con una L2 de 1024 KB, dependiendo de la prueba. La diferencia es de entre un 25% y un 39% en el caso del monoprocesador con una L2 de 512 KB. Además, a partir de 175 peticiones por segundo vemos como el número de peticiones atendidas por segundo decrece. Dado que esto ocurre para las tres configuraciones, es muy probable que este fenómeno suceda porque la red está saturada, lo que provoca que la tasa de peticiones por segundo que llega al servidor decrezca aunque aumentemos la tasa de peticiones por segundo que emite *httperf*.

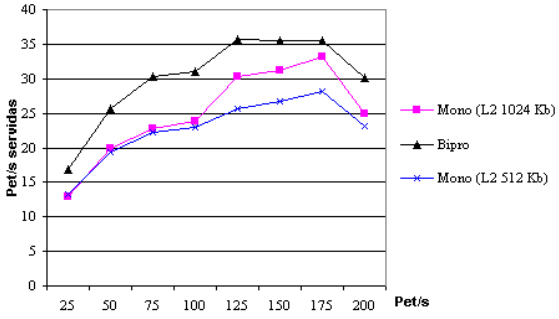

Fig. 1. Peticiones atendidas por segundo en función del número del número de peticiones recibidas por segundo.

B. Análisis detallado de los resultados

1) Estadísticas de la CPU

En las tablas II, III y IV se muestran el número de instrucciones ejecutadas por las CPUs para las arquitecturas evaluadas. En las tres tablas aparecen tanto el número de instrucciones ejecutadas en modo usuario como en modo supervisor. El primer dato interesante es que el número de instrucciones ejecutadas en modo usuario es del orden de 200 veces menor que el número de instrucciones ejecutadas en modo supervisor. Si comparamos las tablas II y IV observamos que el número de instrucciones ejecutadas en modo usuario es prácticamente el mismo en los dos casos, pero se reparte entre los dos procesadores en el caso del biprocesador. No ocurre lo mismo con las instrucciones ejecutadas en modo supervisor, ya que en este caso cada CPU ejecuta por separado aproximadamente el mismo número de instrucciones que el monoprocesador.

Para el caso del monoprocesador con una L2 de 512 KB vemos que el número de instrucciones ejecutadas es en general algo mayor que en el caso de tener una L2 de 1024 KB. Este incremento estaría justificado por una mayor tasa de fallos de caché, como veremos a continuación.

TABLA II

ESTADÍSTICAS DE LA CPU PARA LA ARQUITECTURA MONOPROCESADOR L2 1024 KB

Instrucciones ejecutadas		
Nº pet/s	**Modo Usuario**	**Modo Supervisor**
25	60.672.719	24.674.350.265
50	64.306.550	14.970.932.911
75	64.334.396	11.758.236.372
100	63.820.362	10.066.423.534
125	63.036.580	10.916.560.146
150	63.973.983	12.197.420.294
175	62.724.364	10.824.982.548
200	62.392.239	11.760.974.821

TABLA III

ESTADÍSTICAS DE LA CPU PARA LA ARQUITECTURA MONOPROCESADOR L2 512 KB

Instrucciones ejecutadas		
Nº pet/s	**Modo Usuario**	**Modo Supervisor**
25	60.644.606	24.749.847.588
50	64.532.373	14.331.897.533
75	63.997.248	12.609.442.919
100	63.351.220	11.998.445.012
125	62.783.306	11.524.716.694
150	62.551.989	12.013.025.991
175	62.343.761	12.886.153.678
200	62.092.364	12.774.560.372

TABLA IV

ESTADÍSTICAS DE LA CPU PARA LA ARQUITECTURA BIPROCESADOR

Instrucciones ejecutadas				
Nº pet/s	**Modo Usuario CPU1**	**Modo Supervisor CPU1**	**Modo Usuario CPU2**	**Modo Supervisor CPU2**
25	36.390.337	24.691.231.835	25.184.939	24.702.438.061
50	39.683.545	14.999.605.554	24.870.596	15.014.419.404
75	39.895.103	11.753.254.897	25.331.670	11.767.818.330
100	27.013.742	11.184.858.132	37.279.986	11.174.592.014
125	38.362.852	10.414.023.825	25.219.142	10.427.167.858
150	39.533.435	11.003.628.789	24.228.880	11.018.934.120
175	38.612.952	11.003.658.048	24.098.889	11.618.172.111
200	38.702.460	12.257.364.894	24.062.891	12.272.005.109

TABLA V

ESTADÍSTICAS DE LAS CACHÉS PARA EL MONOPROCESADOR CON L2 DE 1024 KB

	Caché L1	**Caché L2**
Nº Accesos Total	853.019.931	59.572.430
Lecturas	202.450.622	13.986.825
Escrituras	148.847.460	18.408.032
Búsquedas instr.	501.721.849	16.181.040
Fallos lect. (%)	13.457.387 (6'65%)	4.504.676 (32'21%)
Fallos escr. (%)	6.811.658 (4'58%)	2.697.402 (14'65%)
Fallos bús. instr. (%)	16.181.040 (3'23%)	1.280.066 (7'91%)
Tasa de fallos	4'27%	14'24%
Reemplazos	36.424.934	8.436.345
Copy backs	10.996.533	3.731.791
Invalidaciones	24.639	0

TABLA VI

ESTADÍSTICAS DE LAS CACHÉS PARA EL MONOPROCESADOR CON L2 DE 512 KB

	Caché L1	**Caché L2**
Nº Accesos Total	849.761.691	59.572.421
Lecturas	201.546.686	13.947.106
Escrituras	148.236.166	18.369.073
Búsquedas instr.	499.978.839	16.300.024
Fallos lect. (%)	13.386.704 (6'64%)	5.557.981 (39'85%)
Fallos escr. (%)	6.772.486 (4'57%)	3.419.176 (18'61%)
Fallos bús. instr. (%)	16.300.024 (3'26%)	2.504.471 (15'36%)
Tasa de fallos	4'29%	19'27%
Reemplazos	36.398.294	11.401.900
Copy backs	10.956.218	4.596.250
Invalidaciones	60.191	0

TABLA VII

ESTADÍSTICAS DE LAS CACHÉS PARA EL SERVIDOR BIPROCESADOR

	Caché L1 CPU1	**Caché L2 CPU1**	**Caché L1 CPU2**	**Caché L2 CPU2**
Nº Accesos Total	519.494.022	35.505.482	435.295.180	30.508.935
Lecturas	111.218.129	8.355.003	97.576.046	7.167.021
Escrituras	81.950.556	10.293.278	71.646.618	8.896.988
Búsquedas instr.	326.265.992	10.574.259	266.012.222	8.897.637
Fallos lect. (%)	7.838.970 (7'05%)	3.015.892 (36'1%)	6.923.002 (7'09%)	2.649.747 (36'97%)
Fallos escr. (%)	3.721.658 (4'54%)	1.858.609 (18'06%)	3.154.694 (4'4%)	1.529.095 (17'19%)
Fallos bús. instr. (%)	10.574.259 (3'24%)	1.499.542 (14'18%)	8.897.637 (3'34%)	1.203.213 (13'52%)
Tasa de fallos	4'26%	17'95%	4'36%	17'64%
Reemplazos	22.100.601	6.325.123	18.946.037	5.339.169
Copy backs	6.223.927	2.512.316	5.486.995	2.117.483
Invalidaciones	33.774	0	28.784	0

2) Estadísticas de la caché

Los resultados presentados en las tablas V, VI y VII son los correspondientes a las pruebas en que se lanzan 25 peticiones por segundo; los datos del resto de pruebas no varían significativamente. En la tabla V vemos los datos correspondientes a la configuración monoprocesador con L2 de 1024 KB. Comparando los resultados con los de la configuración monoprocesador con L2 de 512 KB (tabla VI), vemos que la diferencia más notable es el aumento en la tasa de fallos de la caché de segundo nivel. También es llamativo el hecho de que el número de invalidaciones de la L1 aumente en un ratio de 2'5 aproximadamente. Este hecho se produce por el aumento en el número de reemplazos (el cual está motivado por el crecimiento de la tasa de fallos), lo cual provoca que sea necesario invalidar más bloques en la caché L1 para mantener la propiedad de inclusión.

Finalmente, observemos los resultados correspondientes a la configuración biprocesador del servidor (tabla VII). En este caso llama la atención de nuevo el mayor número de invalidaciones para las cachés de primer nivel, siendo la explicación la misma que en el caso anterior. En cuanto a las tasas de fallos, para las cachés de primer nivel son prácticamente las mismas que en los casos anteriores; mientras que para las de segundo nivel dicha tasa estaría comprendida entre los valores obtenidos para la configuración monoprocesador con caché L2 de 1024 KB y los obtenidos para la configuración L2 de 512 KB.

3) Errores por timeout de la conexión

Si observamos las tablas II, III y IV de nuevo notamos una tendencia decreciente en el número de instrucciones ejecutadas en modo supervisor conforme el número de peticiones por segundo crece. Este descenso está relacionado con el uso de timeouts para las peticiones. Como vemos en la tabla VIII, conforme el número de peticiones por segundo crece, el número de conexiones que se dan por muertas es mayor. Esto provoca que el número de paquetes que se envían al cliente sea menor, y que por tanto el número de llamadas al sistema operativo para realizar dichos envíos también sea menor.

TABLA VIII

ERRORES DE TIMEOUT EN FUNCIÓN DEL NÚMERO DE PET/S.

Errores por timeout de la conexión			
Nº pet/s	**Mono L2 1024 KB**	**Bipro**	**Mono L2 512 KB**
25	52	47	34
50	89	90	93
75	102	103	104
100	104	105	112
125	128	122	134
150	156	157	169
175	174	194	195
200	200	218	225

VI. CONCLUSIONES

En este artículo hemos introducido un simulador funcional que nos permite realizar la simulación de cargas de trabajo comerciales. También hemos visto cómo nos puede ayudar el simulador a obtener estadísticas sobre la CPU y la jerarquía de memoria del sistema simulado. Sin embargo, existen una serie de limitaciones (modo de ejecución en orden, imposibilidad para simular arquitecturas cc-NUMA, clasificación de los fallos de caché no detallada) que nos impiden realizar un estudio más riguroso de la ejecución de las cargas comerciales.

Por otro lado, en este trabajo hemos empleado *Simics* para determinar bajo qué circunstancias de carga de trabajo un servidor web de contenidos estáticos biprocesador ofrece un rendimiento superior al caso de un monoprocesador.

Como trabajo futuro pretendemos modificar el código fuente del simulador e incluir nuevos módulos que nos permitan ampliar el estudio presentado en este artículo. En concreto, pretendemos ampliar el modelo de cachés del simulador para que permita obtener una taxonomía de los fallos de caché como la propuesta en [1]. Además, estamos trabajando en la caracterización de otros tipos de cargas comerciales, como por ejemplo el benchmark de procesamiento de transacciones en línea TPC-C [13].

REFERENCIAS

[1] M. E. Acacio, J. González, J. M. García y J. Duato. "A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors". *International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, Florida (USA),* April 2002.

[2] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, M. D. Hill, D. A. Wood y D. J. Sorin. "Simulating a $2M Commercial Server on a $2K PC". *IEEE Computer*, February 2003.

[3] A. R. Alameldeen, C. J. Mauer, M. Xu, P. J. Harper, M. M. K. Martin, D. J. Sorin, M. D. Hill y D. A. Wood. "Evaluating Non-deterministic Multi-threaded Commercial Workloads". *5th Workshop Computer Architecure Evaluation using Commercial Workloads (CAECW-02)*, February 2002.

[4] A. R. Alameldeen y D. A. Wood. "Variability in Architectural Simulations of Multi-threaded Workloads". *9th International Symposium on High-Performance Computer Architecture (HPCA-9), Anaheim, CA,* February 2003.

[5] Apache HTTP Server Project. http://httpd.apache.org

[6] Apache Performance Notes. http://httpd.apache.org/docs/misc/perf-tuning.html.

[7] D. E. Culler, J. P. Singh y A. Gupta. "Parallel Computer Architecture: A hardware/software approach". *Morgan Kaufmann Publishers, Inc, San Francisco,* 1999.

[8] C. J. Hughes, V. S. P. Pai, P. Ranganathan y S. V. Adve. "RSIM: Simulating Shared-Memory Multiprocessors with ILP Proccesors". *IEEE Computer*, 35(2):68-76, February 2002.

[9] P. S. Magnusson, M. Christensson, J. Ekilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt y B. Werner. "Simics: A Full System Simulation Platform". *IEEE Computer*, pp. 50-58, February 2002.

[10] D. Mosberger y T. Jin. "httperf: A Tool for Measuring Web Server Performance". *Performance Evaluation Review, Volume 26, Number 3,* pp. 31-37, December 1998.

[11] D. Mosberger y T. Jin. *Httperf man pages.* March 1998.

[12] P. Ranganathan, K. Gharachorloo, S. V. Adve y L. A. Barroso. "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors". *8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pp. 307-318, October 1998.

[13] Transaction Processing Performance Council. TPC Benchmark C, Standard Specification, Revision 5.0, February 2001.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P Singh y A. Gupta. "The SPLASH-2 programs: Characterization and methodological considerations". *22nd International Symposium on Computer Architecture*, pp. 24-36, June 1995.