Sistema Integrado de Simulación de NoCs

Francisco Triviño [1], Francisco J. Andujar [1], Alberto Ros [2], José L. Sánchez [1], Francisco J. Alfaro [1]

Resumen— **La simulación suele ser el método utilizado para evaluar diferentes propuestas sobre componentes de un sistema de computación. Sin embargo, la mayoría de las veces los simuladores se limitan a modelar sólo la parte del sistema en la que se está interesado. No incluir el resto del sistema a la hora de evaluar una determinada configuración de un componente determinado, puede no considerar detalles importantes que acaben influyendo sobre los resultados y conclusiones, que se alcancen a partir del proceso de evaluación. Por esta razón, es cada vez más frecuente que en el ámbito de la arquitectura de computadores se usen simuladores de sistema completo, quizás no tanto en el campo específico de los sistemas de interconexión.**

En este trabajo se describe una herramienta de simulación basada en un sistema completo y orientada a la evaluación de propuestas sobre redes en chip. Es un sistema que integra varias herramientas de simulación ya existentes. En concreto el entorno de simulación está basado en Simics-GEMS extendido con un simulador de redes en chip.

Palabras clave— **Redes en chip, simulación.**

I. Introducción

El número de transistores en los circuitos integrados de los microprocesadores sigue aumentando y con ello la posibilidad de obtener microarquitecturas más complejas que permiten también seguir incrementando su velocidad de procesamiento.

Sin embargo, según ha ido creciendo la densidad de integración ha crecido también la influencia de diversos aspectos, que se prevé mayor cuanto menor sea la tecnología de integración. Esos factores son por ejemplo el consumo o retardos de transmisión [1] y pueden impedir que siga aumentando la frecuencia. Es por ello que en los últimos años están proliferando diseños que buscan seguir aumentando la capacidad de procesamiento pero con otros planteamientos. Una de estas alternativas se apoya en la idea de incluir en un mismo chip varios procesadores más sencillos (cores). Aunque actualmente el número de cores en estos chips es bajo, en un futuro cercano se espera que este valor sobrepase el centenar.

Con un número elevado de cores parece claro que el sistema de interconexión solamente puede realizarse de forma eficiente mediante subsistemas de comunicación escalables. Este es el caso de las redes en chip (Networks on Chip, NoCs) capaces de reducir a valores aceptablemente bajos los tiempos de transmisión de información.

Es importante pues encontrar los modelos de NoCs más adecuados para esta nueva gama de chips multicore (CMPs). Múltiples son las propuestas que se han hecho y a buen seguro seguirán apareciendo, y también diversas las técnicas para evaluarlas. Una de ellas consiste en obtener prototipos [2] donde se toman mediciones sobre un sistema ya existente, pero requiere un gran esfuerzo de desarrollo previo que hace inviable en muchos casos dicha posibilidad. También se puede hacer uso de modelos analíticos pero éstos no son adecuados para sistemas de gran complejidad. Por último, se suele recurrir a la simulación para analizar y comparar la efectividad de una determinada propuesta con otras similares. Los sistemas simulados son los más ampliamente utilizados y permiten el desarrollo de distintos niveles de detalle. En estos casos, es deseable que las simulaciones sean lo más fieles a la realidad, siempre que se pueda realizar en un tiempo razonable.

El diseño de NoCs conlleva importantes elecciones tales como la topología, algoritmos de encaminamiento, políticas de arbitraje, posibles mecanismos para ofrecer soporte de calidad de servicio, etc. Tan importante como describir bien el sistema de interconexión en el simulador es tener bien caracterizada la carga de trabajo a la que será sometido.

Muchos trabajos de investigación han realizado dichos estudios teniendo en cuenta que la carga generada por las aplicaciones puede ser modelada mediante tráfico sintético que generalmente consiste en utilizar frecuencias constantes de generación de mensajes, distribuciones uniformes de destinos y patrones fijos de tráfico. Aunque esas cargas representan una aproximación a la carga generada por aplicaciones reales bien es cierto que cuando el diseñador debe realizar las elecciones finales se debe llevar a cabo un estudio más detallado y preciso.

Por tanto, los sistemas de comunicaciones tienen que ser diseñados teniendo en cuenta cómo van a ser usados por las aplicaciones paralelas y distribuidas que se ejecutarán sobre ellos al igual que se hace para el diseño de los procesadores.

Utilizar cargas reales implica buscar aplicaciones que sean representativas. En el caso de los CMPs no parece conveniente seguir considerando las aplicaciones paralelas clásicas de memoria compartida usadas en los grandes sistemas multiprocesador. Un ejemplo de aplicaciones de este tipo las podemos encontrar en el benchmark SPLASH-2 [3] cuyas aplicaciones están muy orientadas a computación de altas prestaciones (HPC) y programas gráficos que utilizan algoritmos ya en desuso. OMP2001 [4] carece de algunas de las técnicas de paralelización mas actuales tales como el modelo “Pipeline” [5] o el modelo “productor-consumidor” [6]. Otro ejemplo son las SPEC CPU2006, que es un conjunto de programas que implementa código serie y que no están pensados

[1]Dpto. de Sistemas Informáticos, Univ. Castilla-La Mancha, e-mail: {ftrivino, fandujar, jsanchez, falfaro}@dsi.uclm.es

[2]Dpto. de Ingeniería y Tecnología de Computadores, Univ. de Murcia, e-mail: a.ros@ditec.um.es

para evaluar máquinas paralelas.

En cambio, el repositorio PARSEC [7] ha sido diseñado para recoger un largo y variado conjunto de aplicaciones que han sido correctamente paralelizadas con diferentes técnicas. Se trata de 9 aplicaciones y 3 kernels que cubren un amplio conjunto de aplicaciones de diferentes dominios tales como visión por computador, procesamiento multimedia, servidores empresariales, aplicaciones físicas, etc. Otra de las características a tener en cuenta es la posibilidad de adecuar la carga de trabajo de cada aplicación mediante el uso de hasta seis tipos de tamaños de carga diferentes.

En este trabajo se presenta una herramienta de simulación para evaluar de la forma más realista posible propuestas sobre NoCs. Se ha elegido un conjunto de simuladores de diferente tipo que se integran en una única herramienta de simulación de sistema completo que presenta al software una arquitectura hardware real. Este sistema ha sido extendido con un simulador de NoCs para poder evaluar diversas propuestas relacionadas directamente con estas redes.

Este artículo está organizado de la siguiente manera: en la sección II se resumen las características más importantes de los simuladores utilizados. En la sección III se describe el proceso de integración de esos simuladores para obtener la herramienta completa. Finalmente, en la sección IV se presentan las conclusiones y trabajo futuro.

II. ENTORNO DE SIMULACIÓN

Como se ha indicado anteriormente, una gran ventaja de los simuladores es que se puede elaborar con gran nivel de detalle un determinado modelo. A través de múltiples parámetros se pueden obtener distintas configuraciones de forma rápida y sencilla. Sin embargo, en muchos casos no se modela un sistema de computación completo sino sólo una parte del mismo. Es el caso de las NoCs en particular o las redes de interconexión en general. No incluir el resto del sistema a la hora de evaluar una determinada configuración de red puede no considerar detalles importantes que acaben influyendo sobre los resultados y conclusiones que se alcancen a partir del proceso de evaluación. Por esta razón, es cada vez más frecuente que en el ámbito de la arquitectura de computadores se usen simuladores de sistema completo, quizás no tanto en el campo específico de los sistemas de interconexión.

Se describe en este trabajo una herramienta de simulación basada en sistema completo y orientada a la evaluación de propuestas sobre NoCs. Se trata de un sistema de simulación basado en herramientas existentes, y en el que lo más importante es la integración que se ha hecho con ellas. En concreto, el entorno de simulación está basado en Simics [8] extendido con GEMS [9] e integrando el simulador de NoC Noxim [10]. GEMS proporciona el nivel de detalle deseado en cuanto a la jerarquía de memoria, mientras que Noxim modela la red de interconexión.

A continuación se describen brevemente las partes básicas, y en la sección III se detalla la integración realizada. Hay que señalar que la integración es altamente independiente del modelo de red incorporado y por tanto se requerirían muy pocos cambios para considerar otro simulador de red diferente.

A. Simics

La plataforma Virtutech Simics [8] es una plataforma de simulación de un sistema completo capaz de simular gran variedad de arquitecturas diferentes que modelan funcionalmente todos los elementos hardware del sistema. La herramienta permite ejecutar de manera simulada sistemas operativos y sobre éstos aplicaciones comerciales. Se trata de un simulador dirigido por eventos con resolución máxima de un ciclo de reloj que permite modificar la máquina simulada ofreciendo estadísticas que ayudan a comprender el funcionamiento del sistema. Es utilizado principalmente para tareas de investigación en arquitectura de computadores, desarrollo de sistemas operativos y simulación conjunta de hardware y software.

Simics simula hasta un total de diez familias de procesadores diferentes, algunas de ellas son: UltraSparc II/III, x86/64, Alpha, PowerPC, etc. Sobre dichas arquitecturas es posible la instalación de cualquier sistema operativo siempre y cuando ofrezca soporte para la arquitectura finalmente elegida. Así, es posible ejecutar aplicaciones reales que generalmente incluyen algún tipo de benchmark adecuado para la evaluación del sistema.

B. GEMS

GEMS [9] (General Execution-driven Multiprocessor Simulator) está compuesto por un conjunto de módulos desarrollados exclusivamente para Simics. Una de las características de GEMS es que hace posible la simulación de sistemas multiprocesadores de forma más detallada. Básicamente está formado por dos módulos (Figura 2), Ruby que implementa el modelo de la jerarquía de memoria y Opal que permite simulación con ejecución fuera de orden.

GEMS Posibilita el modelado de diferentes arquitecturas, desde monoprocesadores hasta multiprocesadores como SMPs, CC-NUMAs y CMPs, donde es posible la ejecución de varios hilos de forma simultánea (Figura 1).

Ruby es uno de los principales módulos que implementa GEMS y su principal objetivo es simular la jerarquía de memoria, esto es: caches, controladores de memoria, bancos de memoria principal e incluso la red que interconecta todos estos dispositivos tal y como se muestra en la Figura 2 (Basada en [9]).

Ruby ofrece también la posibilidad de configurar distintas relaciones de frecuencia entre los procesadores, jerarquía de memoria y red de interconexión. Esto es una característica importante si se tiene en cuenta que en las máquinas actuales la velocidad de la memoria tiende a ser cada vez más lenta con respecto a la velocidad del procesador.

También se pueden usar protocolos de coherencia,

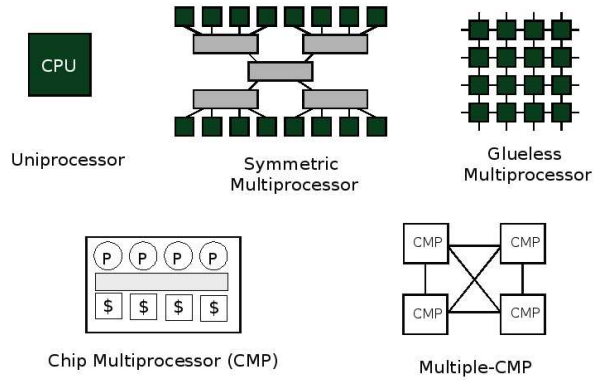

Fig. 1. Arquitecturas modeladas por GEMS.

destinados a preservar la integridad de los datos repartidos sobre los diferentes niveles de la jerarquía de memoria. Ruby ofrece la posibilidad de modelar diferentes protocolos de coherencia snoopy y basados en directorio.

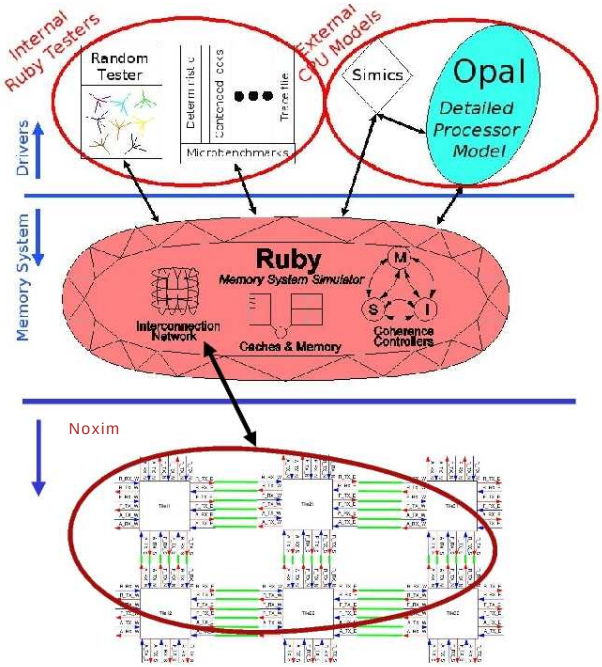

Fig. 2. Componentes de GEMS extendidos por Noxim.

C. Noxim

La red de interconexión es la capa de comunicación entre los diferentes componentes del sistema. Ruby la utiliza para comunicar los controladores de cache y memoria mediante una red sencilla que simula todas las comunicaciones. Toda la comunicación intra-chip e inter-chip se maneja como parte de esta interconexión, aunque cada enlace individual puede tener distintos parámetros de latencia y ancho de banda. En este trabajo, se ha sustituido esta red por un simulador de redes en chip llamado Noxim, cuyo núcleo de simulación se apoya en SystemC (Figura 2). Este simulador ofrece diferentes posibilidades de configuración y tiene en cuenta la mayoría de los aspectos de una red de interconexión dentro del chip.

Por tanto, Noxim ofrece la posibilidad de cambiar importantes elecciones en el diseño de una NoC determinada tales como la estrategia de encaminamiento, distribución de tareas en nodos, políticas de arbitraje, diferentes soportes para calidad de servicio, tamaño de los buffers, etc. Modela topologías malla 2D, actualmente la opción ampliamente considerada para las NoCs. Noxim usa wormhole como técnica de conmutación y el protocolo ABP como técnica de control de flujo. Incorpora canales virtuales y varios algoritmos de encaminamiento deterministas y adaptativos.

Otra de las características que debe tener toda herramienta de simulación es que debe ofrecer estadísticas de todo lo sucedido. Así, el simulador Noxim, como resultado de una simulación, obtiene valores de las métricas habituales como: paquetes/flits recibidos, productividad global y por nodo, latencia media y máxima, grado de utilización de los enlaces, etc.

Como ya se ha mencionado, Noxim se apoya en SystemC para modelar una red de interconexión de la forma más real posible. SystemC es un lenguaje de descripción de hardware similar a VHDL y Verilog y su principal característica radica en el modelado de sistemas a nivel de comportamiento. Los objetos descritos bajo este lenguaje son capaces de comunicarse durante una simulación de tiempo real utilizando señales de cualquier tipo. Un sistema en SystemC está formado por un conjunto de módulos que describen cierta funcionalidad y se comunican con otros módulos a través de canales o eventos.

III. Integración del sistema

Gran parte del trabajo desarrollado para la integración de los diferentes simuladores ha sido la inclusión de un nuevo modelo de redes de interconexión en el simulador Ruby. Como ya se ha mencionado anteriormente, Ruby incorpora una red sencilla para simular las latencias producidas por la transmisión de la información a través de la red. Esta red ha sido sustituida por un simulador más detallado cuyo núcleo se apoya en SystemC.

Ruby utiliza un modelo de simulación dirigido por eventos, y está basado en una cola global cuyo funcionamiento no es estrictamente FIFO sino que funciona por prioridades auto-ordenadas. Cuando se produce un determinado evento, se analiza su tipo y se ejecuta el código asociado a dicho evento.

Para poder conseguir integrar el simulador de red Noxim en Ruby se ha tenido que desarrollar una clase llamada noximNetwork que cumple con las reglas de implementación de los eventos anteriormente descritos, de forma que, se incluye un evento de tipo noximNetwork en dicha cola de espera. Cuando es el turno de procesar dicho evento, éste ejecuta su método wakeup() común a todos los tipos de eventos y que dependiendo del tipo de evento hará que se ejecuten unas acciones u otras. El pseudocódigo implementado en dicho método para

la clase noximNetork se muestra en la Figura 3. Una vez finaliza su ejecución se introduce de nuevo el evento noximNetwork en la cola global de eventos.

```
Inicio

   Para cada puerto de entrada
      Para cada mensaje en el puerto
         Extraer_mensaje()
         Transformar_mensaje()
         Inyectar_en_noxim(mensaje)
         Almacenar_id(mensaje)
      Fin para
   Fin para

   Activar_señal_reloj_noxim()
   Ejecutar_ciclo(1)
   Desactivar_señal_reloj_noxim()

   Para cada elemento de proceso
      Para cada mensaje recibido
         Extraer_mensaje()
         Recuperar_id(mensaje)
         Inyectar_en_ruby(mensaje)
      Fin para
   Fin para

   Cola_global <- Objeto noximNetwork

Fin
```

Fig. 3. Pseudocódigo del método wakeup().

El pseudocódigo de la Figura 3 consta de tres fases bien diferenciadas:

1. Se obtienen los mensajes que Ruby debe procesar y enviar a través de la red de interconexión. Los mensajes deben ser transformados a paquetes que acepta Noxim para ser inyectados en la red.

2. Se ejecuta un ciclo de reloj de Noxim.

3. Se examinan los paquetes recibidos desde Noxim, se transforman a mensajes de Ruby y se envían a la componente adecuada de Ruby. La última acción es inyectar de nuevo el evento noximNetwork en la cola global de eventos para dar continuidad al simulador Noxim.

Para comprender mejor el funcionamiento de la herramienta, en la Figura 4 se ofrece una visión global del sistema completo. Cada vez que Simics realiza una transacción de memoria(1) (Load o Store) se comunica con la interfaz encargada de recibir peticiones y pasarlas a Ruby para que se ejecuten en el sistema de memoria. El Driver inmediatamente envía una señal de espera infinita a Simics(2). Mientras, se encarga de modelar dicha transacción en el CMP simulado(3). Es decir, como consecuencia de una transacción de memoria se producirán una serie de mensajes en la red entre diferentes elementos de la jerarquía de memoria que han de ser simulados. Una vez finaliza la simulación de la transacción solicitada se devuelve el control al simulador Simics para que continúe con el flujo de transacciones de memoria(4)(5).

Una interesante característica que se ha añadido en este trabajo es la posibilidad de recoger el tráfico generado en la red de interconexión mediante la obtención de trazas. Estas trazas están formadas por mensajes que se envían y reciben entre los diferentes dispositivos que forman el sistema de comunicación modelado.

Generalmente las trazas de tráfico tradicionales marcan los mensajes en la red con un determinado tiempo, y por tanto, cuando se alcance dicho tiempo los mensajes serán enviados a sus correspondientes destinos a través de la red. Este sistema presenta ciertas limitaciones ya que esos tiempos dependen de las condiciones o situaciones en las que se generó la traza. Para solucionar esto, se puede utilizar tráfico auto-relacionado. Un paquete A es generado en un elemento de proceso sólo después de que en éste se haya recibido otro determinado paquete B; del cual decimos que depende el paquete A. Dicho de otro modo, en la traza se indica para cada mensaje si su inyección en la red desde un nodo dado depende o no de la llegada de otro mensaje a dicho nodo. Por ejemplo, un nodo no enviará un mensaje correspondiente a un bloque de datos hasta que no reciba un mensaje solicitando dichos datos.

En la Figura 5 se observan dos tipos de dependencias. En la primera intervienen los dos primeros mensajes: el core 0 solicita un conjunto de datos al core 1, como consecuencia se produce un mensaje de tipo GETS. El core 1 procesa dicho mensaje y tras 10 ciclos envía copia de los datos al core 1. El mensaje DATA quedará registrado en la traza con una dependencia del mensaje GETS y solamente será inyectado en la red una vez pasen 10 ciclos después de la recepción del mensaje GETS. El segundo tipo de dependencia queda reflejado con los dos últimos mensajes: una vez finalizado el flujo de datos anterior en el core 0, se produce un mensaje de tipo UNBLOCK que posibilita nuevos flujos de información, por lo tanto el segundo tipo de dependencia se produce entre el mensaje UNBLOCK y el siguiente GETS. Dicho de otro modo, pasados 40 ciclos del envío del mensaje UNBLOCK el core 0 puede continuar con el envío de un nuevo GETS. Los únicos mensajes independientes estarán formados por el primer mensaje emitido en cada core.

En el ejemplo de la Figura 6 se pueden observar las primeras líneas de una traza obtenida en una determinada simulación. Cada línea es un mensaje que circula por la red y está compuesto por los campos expresados en la Tabla I. Se observa, por ejemplo, que el mensaje con Id 4 tiene una dependencia con el mensaje con Id 0. A su vez, el mensaje con Id 8

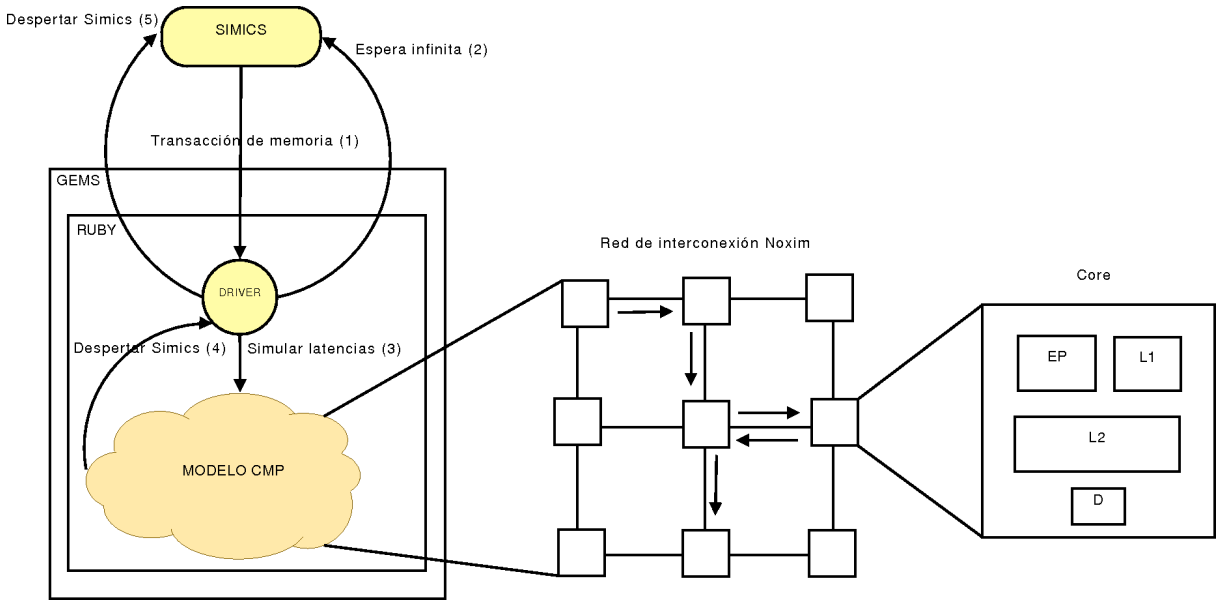

Fig. 4. Funcionamiento global del sistema completo.

depende del mensaje 4, el 9 depende del 8, y de la misma forma para el resto de mensajes. En este caso se trata de una petición de datos para lectura que realiza la cache L1 del nodo 0. La petición es dirigida a la cache L2 del nodo 0, el cual redirecciona la petición al directorio del nodo 2, donde se encuentra la información de los datos solicitados. Posteriormente, se envían los datos a la L2 del nodo 0 mediante un mensaje DATA_EXCLUSIVE en el mensaje 8 y por último, en el mensaje 9, la L2 del nodo 0 envía copia a la L1 del mismo nodo satisfaciendo la petición de datos inicial.

Un valor -1 en el último campo del mensaje indica que ese mensaje no depende de ningún otro. Además, este tipo de mensajes sólo se dan al principio, por lo que todos los mensajes posteriores son dependientes.

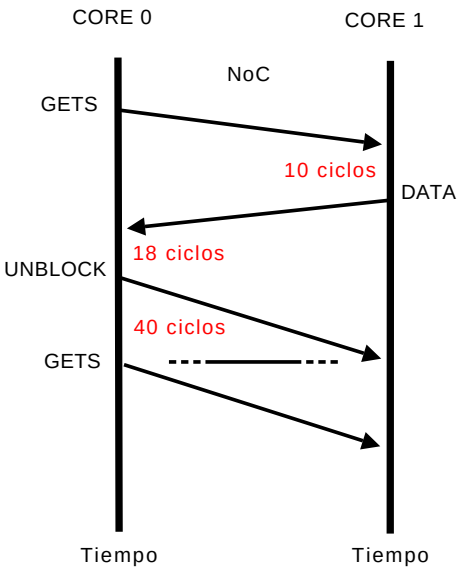

Fig. 5. Dependencias entre mensajes.

TABLA I
CAMPOS DE LOS MENSAJES.

Campo	Definición
ID	Identificador del mensaje
OCore	Core origen del mensaje
OComponente	Componente origen del mensaje
DCore	Core destino del mensaje
DComponente	Componente destino del mensaje
Tipo	Tipo de mensaje
Tamaño	Tamaño en flits del mensaje
TComputación	Tiempo de computación
Dependencia	Identificador del mensaje del que depende

Para conseguir este modo de funcionamiento, se ha tenido que modificar el código fuente en el simulador Ruby para que registre todos los mensajes que circulan por la red. Así, tras realizar una simulación, se obtendrá la traza asociada que contiene el tráfico generado en el CMP modelado. Por otra parte se ha tenido que adaptar el simulador Noxim para dar soporte a este tipo de trazas auto-relacionadas.

La decisión de generar este tipo de trazas viene dada por el hecho de que el tiempo necesario para realizar simulaciones con el sistema completo es bastante elevado. Cuando se tiene que realizar una gran cantidad de pruebas esto puede suponer semanas, o incluso meses, para obtener unos determinados resultados. Utilizando directamente las trazas con el simulador de red, Noxim en este caso, los tiempos de simulación se reducen de una manera considerable, de un orden de magnitud temporal. El planteamiento podría ser usar directamente el simulador de red y las trazas para encontrar las configuraciones que sean posiblemente más adecuadas, y usar el simulador de sistema completo para obtener los resultados definitivos de forma más precisa.

Como consideración final decir que el uso de trazas

```
0 0 L1Cache 0 L2Cache GETS 8 3 -1
1 1 L1Cache 1 L2Cache GETS 8 3 -1
2 2 L1Cache 0 L2Cache GETS 8 3 -1
3 3 L1Cache 1 L2Cache GETS 8 3 -1
4 0 L2Cache 2 Directory GETS 8 2 0
5 1 L2Cache 2 Directory GETS 8 2 1
6 1 L2Cache 3 Directory GETS 8 2 3
7 0 L2Cache 3 Directory GETS 8 2 2
8 2 Directory 0 L2Cache DATA_EXCLUSIVE 72 158 4
9 0 L2Cache 0 L1Cache DATA_EXCLUSIVE 72 4 8
10 0 L1Cache 0 L2Cache UNBLOCK_EXCLUSIVE 8 1 9
11 3 Directory 1 L2Cache DATA_EXCLUSIVE 72 162 6
12 0 L1Cache 0 L2Cache GETS 8 8 9
13 0 L2Cache 2 Directory UNBLOCK_EXCLUSIVE 8 4 10
14 2 Directory 1 L2Cache DATA_EXCLUSIVE 72 160 5
15 3 Directory 0 L2Cache DATA_EXCLUSIVE 72 160 7
16 1 L2Cache 1 L1Cache DATA_EXCLUSIVE 72 4 14
17 1 L1Cache 1 L2Cache UNBLOCK_EXCLUSIVE 8 1 16
18 1 L2Cache 3 L1Cache DATA_EXCLUSIVE 72 4 11
19 1 L1Cache 1 L2Cache GETS 8 6 16
20 3 L1Cache 1 L2Cache UNBLOCK_EXCLUSIVE 8 1 18
21 3 L1Cache 1 L2Cache GETS 8 3 18
22 0 L2Cache 1 Directory GETS 8 2 12
...          ...          ...          ...
```

Fig. 6. Primeras líneas de una traza.

con el simulador Noxim ofrece resultados aproximados puesto que se da la posibilidad de que la traza haya sido generada bajo condiciones de red diferentes. Por tanto, resulta necesario comprobar que los resultados ofrecidos por los dos modos de funcionamiento no ofrecen diferencias significativas. Mediante el uso de métodos estadísticos se pueden evaluar ambos modos de funcionamiento y asegurar la compatibilidad en los resultados.

IV. Conclusiones

En este trabajo se ha mostrado como es posible utilizar una herramienta de simulación basada en un sistema completo para la evaluación de propuestas sobre NoCs utilizando para ello Simics y GEMS.

En muchos casos de la literatura no se modela un sistema de computación completo sino sólo una parte del mismo. No incluir el resto del sistema a la hora de evaluar una determinada configuración de red puede no considerar detalles importantes que acaben influyendo sobre los resultados y conclusiones que se alcancen a partir del proceso de evaluación.

También, se ha descrito cómo realizar la integración de un simulador de NoCs específico en el sistema Simics-GEMS y deja abierta la posibilidad de considerar otros simuladores de red diferentes, puesto que el proceso de integración lo permite, al ser altamente independiente del modelo de red incorporado.

Por otra parte, se ha visto la capacidad de modelado que ofrece la herramienta de simulación, que permite infinidad de configuraciones tanto hardware como software con un alto nivel de detalle. La principal desventaja de simular el sistema completo es que los tiempos de simulación son muy elevados cuando se tienen que obtener gran cantidad de pruebas.

En este trabajo se ha detallado una posible solución que se basa en el uso de trazas auto-relacionadas para su uso sólo con el simulador de red, Noxim en este caso, para encontrar las configuraciones de red óptimas. No obstante, a la hora de obtener resultados definitivos se requiere el uso de la herramienta completa.

En el momento de redactar este documento, se estaba indicando un estudio para comprobar, mediante técnicas estadísticas, si los resultados obtenidos con el sistema completo ofrecen diferencias significativas o no con los resultados obtenidos en el simulador de red usando trazas. En el caso de que no las haya, la gran mayoría de pruebas se podrán realizar sólo con el simulador de red, lo cual reduce considerablemente el tiempo empleado por las simulaciones.

Agradecimientos

Este trabajo ha sido cofinanciado por el Ministerio de Educación y Ciencia, y por fondos FEDER de la Comisión Europea, con las subvenciones "Consolider Ingenio–2010 CSD2006–00046" y "TIN2006–15516–C04–02" respectivamente; y por la Consejería de Educación y Ciencia de Junta de Comunidades de Castilla–La Mancha con el proyecto PCC08–0078–9856.

Referencias

[1] Vikas Agarwal, M.S. Hrishikesh, Stephen W. Keckler, and Doug Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures," 2000.

[2] Sriram Vangal, "An 80-tile sub-100-w teraflops processor in 65-nm cmos,"l. ieee journal of solid-state circuits," *l. IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, 2008.

[3] Jaswinder Pal Singh, Anoop Gupta, Moriyoshi Ohara, Evan Torrie, and Steven Cameron Woo, "The SPLASH-2 programs: Characterization and methodological considerations," *Computer Architecture, International Symposium on*, vol. 0, pp. 24, 1995.

[4] Vishal Aslot and Rudolf Eigenmann, "Performance characteristics of the SPEC OMP2001 benchmarks," in *in 3rd European Workshop on OpenMP, EWOMP'01*, 2001.

[5] Wei-Keng Liao, Alok Choudhary, Donald Weiner, and Pramod Varshney, "Performance evaluation of a parallel pipeline computational model for space-time adaptive processing," *J. Supercomput.*, vol. 31, no. 2, pp. 137–160, 2005.

[6] Lisa Higham and Jalal Kawash, "Critical sections and producer/consumer queues in weak memory systems," *Parallel Architectures, Algorithms, and Networks, International Symposium on*, vol. 0, pp. 56, 1997.

[7] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, "The PARSEC Benchmark Suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[8] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[9] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[10] Simulador Noxim., *http://noxim.sourceforge.net/*.

[11] David E. Culler, Anoop Gupta, and Jaswinder Pal Singh, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.