# Un Simulador de Redes de Interconexión Dirigido por Ejecución para Sistemas DSM: EDINET

J. Flich, P. López, M. P. Malumbres, J. Duato

*Resumen*— **En la evaluación de redes de interconexión de multiprocesadores con memoria distribuida se suelen utilizar cargas sintéticas o cargas generadas a partir de trazas de ejecución. Aunque estas cargas representan una aproximación de la carga generada por aplicaciones reales en las primeras etapas de la evaluación, en las decisiones finales del diseñador, un estudio de evaluación más preciso debe ser llevado a cabo. En este artículo, describimos una nueva herramienta de simulación dirigida por ejecución para evaluar redes de interconexión de multiprocesadores con memoria distribuida usando la carga generada por aplicaciones reales. Esta herramienta se basa en dos simuladores: un simulador de memoria dirigido por ejecución y un simulador de redes de interconexión. Usando esta herramienta, hemos desarrollado un modelo de memoria NCC-NUMA y hemos obtenido algunos resultados de simulación de la suite SPLASH-2, para diferentes algoritmos de encaminamiento.**

## I. Introducción

En los últimos años muchos trabajos de investigación han evaluado redes de interconexión de multiprocesadores de memoria distribuida. En estos estudios usualmente se asume que la carga generada por las aplicaciones puede ser modelada usando cargas sintéticas. Así pues, es frecuente usar frecuencias constantes de generación de mensajes para todos los nodos y distribución uniforme de destinos de los mensajes con o sin localidad y patrones fijos de tráfico (complemento, bit-reversal, ...). Aunque estas cargas pueden ser usadas como una aproximación de la carga generada por las aplicaciones reales, cuando el diseñador debe realizar las elecciones finales, debe llevarse a cabo un estudio de evaluación más preciso.

Otros estudios sobre redes de interconexión se han basado en simulaciones dirigidas por trazas [21], [18]. En este caso, la carga es generada por otra herramienta que simula la aplicación real y genera un fichero de trazas [20], [4]. Sin embargo, aunque las simulaciones dirigidas por trazas son adecuadas para simulaciones de máquinas uni-procesador, en multiprocesadores, debido a la naturaleza no determinista del paralelismo, este método no garantiza la exactitud de los resultados obtenidos. Por ejemplo, algunos mensajes no deben generarse hasta que otro u otros hayan llegado a sus destinos, y esto depende de la red de interconexión, que no esta siendo simulada cuando el fichero de trazas se genera.

La herramienta precisa para conseguir resultados de evaluación exactos es el simulador dirigido por ejecución [7], [14]. En estos simuladores, una aplicación se ejecuta en el procesador host, ejecutando su código nativo, y se insertan llamadas especiales en el código original para instrumentar los eventos que desean ser analizados. Estos eventos son enviados como peticiones al simulador. En nuestro caso, los eventos son las primitivas send/receive, y el simulador es el simulador de redes de interconexión. Adicionalmente, se requiere un simulador de aplicaciones paralelas para ejecutar el código paralelo en un sólo procesador.

Por otra parte, los multiprocesadores de memoria distribuida pueden utilizar el modelo de paso de mensajes o el modelo de memoria compartida. En el primer caso, los mensajes son generados cuando los procesos ejecutan directamente una primitiva send/receive. Usualmente, en estas máquinas se desarrollan las aplicaciones utilizando el modelo estándar MPI [17] o PVM [8]. Cuando se emplea el modelo de memoria compartido, los procesos que están ejecutándose intercambian información por medio de variables compartidas, usando la red de interconexión para acceder a posiciones de memoria remotas [9] o para soportar el protocolo de coherencia de caches [11]. Estos sistemas son conocidos como DSMs [16]. Los DSMs son muy populares en la actualidad debido al modelo de programación de memoria compartido que incorporan y a su buena escalabilidad. Desde el punto de vista de la red de interconexión, los mensajes son directamente enviados por el hardware de los DSMs como consecuencia de un acceso a memoria remoto o una orden de coherencia, reduciendo considerablemente la sobrecarga al eliminar la llamada al sistema que tiene que ser invocado en los multicomputadores [12].

Muchas máquinas comerciales incorporan el modelo de programación de memoria compartida y/o el modelo por paso de mensajes. Por ejemplo, el Origin 2000, HP/Convex Exemplar, Sequent NUMA-Q y HAL S1 son algunas de estas arquitecturas. Otras máquinas DSM han sido desarrolladas en laboratorios de investigación como el Stanford DASH[11], FLASH[10] y MIT Alewife[1].

Tomando en cuenta la creciente popularidad de los sistemas DSM, parece lógico que la evaluación de redes de interconexión se realice considerando el tráfico generado por aplicaciones de memoria compartida. En este artículo, presentamos una nueva herramienta de simulación, denominada EDINET (Execution-Driven Interconnection NETwork simulator). Este simulador modela la ejecución de una aplicación de memoria compartida en un sistema DSM, usando la carga generada dinámicamente por la misma para analizar la red de interconexión, lo cual permite seleccionar los parámetros de la red de interconexión más adecuada que permiten reducir el tiempo de ejecución de la aplicación.

En particular, estamos interesados en analizar la influencia del encaminamiento adaptativo, que ha resultado ser muy interesante para cargas sintéticas.

Sin embargo, el simulador EDINET también permite simular con detalle el subsistema de memoria de la máquina. Por lo tanto, usando esta herramienta, la influencia de algunas elecciones de diseño concernientes a ambos subsistemas pueden ser analizados simultáneamente.

El simulador EDINET esta compuesto por dos simuladores. El primero es el Limes[14], un simulador dirigido

Grupo de Arquitecturas Paralelas, Depto. Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, Camino de Vera s/n, 46071 - Valencia, SPAIN. E-mail: jflich@gap.upv.es .

Este trabajo ha sido subvencionado por el proyecto de la CICYT TIC97–0897–C04–01.

por ejecución que permite la ejecución de aplicaciones paralelas y modela el sistema de memoria. El segundo es el simulador de redes de interconexión que ya hemos usado en muchos estudios de evaluación [5], [15].

Este artículo está organizado como sigue. En la sección II y III se expone el simulador de memoria Limes y el simulador de redes de interconexión. La sección IV presenta el nuevo simulador EDINET, detallando algunos de sus aspectos internos. Como ejemplo de aplicación, la sección V muestra resultados de simulación obtenidos con el simulador sobre una arquitectura con un modelo de memoria NCC-NUMA [19](Non Cache Coherent Non-Uniform Memory Access). Finalmente, se presentan algunas conclusiones en la sección VI.

## II. Limes

El simulador Limes[14] ha sido desarrollado por Davor Magdic en la Universidad de Belgrado. Es una herramienta que permite la simulación de un multiprocesador sobre un sistema uni-procesador, ejecutándose en un PC con un i486 o CPU superior con el sistema operativo Linux. La figura 1 muestra el diagrama de bloques del Limes. La aplicación paralela es un programa creado para ser ejecutado en un sistema con N procesadores. La aplicación consiste en N hilos de ejecución que operan en paralelo. El código está escrito en el lenguaje de programación C con las macros ANL[13], que lo hace portable a máquinas multiprocesador reales.

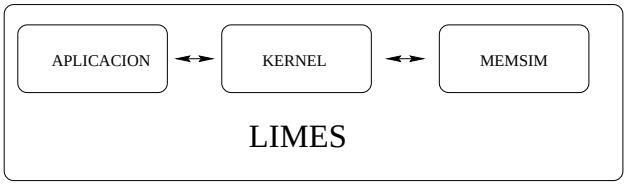


Fig. 1. Diagrama de bloques del Limes

El simulador de memoria (denominado *MemSim*) modela la arquitectura que yace bajo los procesadores, y simula las peticiones de memoria. Se puede implementar cualquier modelo de memoria que se desee estudiar. Por defecto, Limes viene con un modelo PRAM(perfect RAM)[6] y algunos modelos *snoopy* basados en bus.

Finalmente, está el kernel del simulador. Está ubicado entre las aplicaciones y el simulador de memoria. El kernel del simulador captura solamente los eventos de interés. Pueden ser referencias a datos compartidos, primitivas de sincronización, o todos los accesos a memoria. Es responsabilidad del kernel detener y re-ejecutar los hilos según se van generando eventos, y llamar al simulador de memoria con la descripción de los eventos en los tiempos correctos. La aplicación paralela se ejecuta en el procesador local, insertándose llamadas especiales en el código original de la aplicación para instrumentar los eventos que la aplicación genera. Este proceso se llama *augmentation* y es realizado en el proceso de compilación de la aplicación.

Finalmente la aplicación, el kernel y el modelo de memoria deseado se enlaza en un ejecutable.

En Limes se asume que cada instrucción que no genera un evento al simulador de memoria dura un ciclo. Esta aproximación puede ser fácilmente justificada teniendo en cuenta que las CPU de hoy en día son segmentadas.

En EDINET, desarrollaremos un simulador de memoria para cada arquitectura DSM que queramos estudiar. Más adelante en este artículo presentaremos un simulador de memoria para la arquitectura NCC-NUMA (Non Cache-Coherent Non-Uniform Memory-Access).

## III. Simulador de Redes de Interconexión (NetSim)

El simulador de redes de interconexión ha sido ampliamente utilizado para evaluar diferentes aproximaciones de diseño usando cargas sintéticas [5], [15]. El simulador modela la red a nivel de flit. Cada nodo de la red consta de un procesador, su memoria local, un encaminador, un conmutador y varios canales. La memoria se comunica con el conmutador a través de cuatro puertos.

El encaminador decide el canal de salida para un mensaje en función del nodo destino, el nodo actual y el estado de los canales de salida. El encaminador solamente puede procesar una cabecera de mensaje por unidad de tiempo. Los mensajes son atendidos con una política *round-robin* (incluyendo aquellos mensajes generados por el procesador local). Cuando un mensaje llega al encaminador pero no puede ser encaminado porque todos los canales de salida están ocupados, debe esperar en la cola de entrada hasta que llegue su próximo turno. El conmutador es un *crossbar*, permitiendo que múltiples mensajes lo atraviesen simultáneamente sin interferencia. El conmutador es configurado por el encaminador cada vez que un encaminamiento ha sido realizado de forma satisfactoria.

Los canales físicos pueden ser multiplexados en varios canales virtuales. Los canales virtuales son asignados al enlace físico usando un esquema de arbitraje round-robin. Cada canal virtual tiene un buffer asociado, estando dividido en dos mitades, una asociada con el puerto de salida del conmutador, y la otra asociada con el puerto de entrada del conmutador del siguiente nodo. El tamaño de estos buffers es mantenido constante e igual a 4 flits.

Los tiempos de encaminamiento, conmutador y retardos de canal se suponen que son iguales a 1 ciclo de reloj.

## IV. Edinet

El simulador Edinet básicamente consiste en interconectar el Limes y el simulador de redes de interconexión. En particular, el simulador de redes de interconexión trabaja como un proceso hijo del Limes. El simulador de memoria enviará peticiones al simulador de redes para simular el avance de los mensajes a través de la red de interconexión cuando sea necesario. El simulador de redes avisará al simulador de memoria cuando el mensaje llegue a su nodo destino. Todo esto es implementado usando tubos (*pipe channels*) entre los dos simuladores. La figura 2 muestra el diagrama de bloques de Edinet.

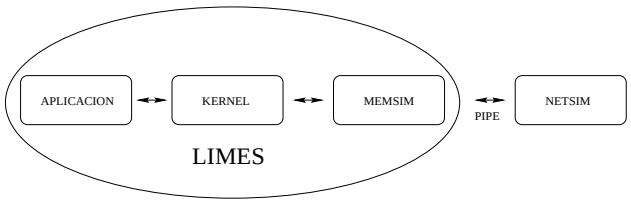


Fig. 2. Estructura de EDINET

El subsistema de memoria (caches, directorios, ...) es

modelado en el simulador de memoria, dejando solamente el comportamiento de la red al simulador de redes. Así, por ejemplo, en un modelo de memoria con caches, solamente los fallos de cache invocarán al simulador de redes.

El simulador de memoria controla el simulador de redes usando las siguientes órdenes:

- *InitNetSim*. Este orden indica al NetSim que debe empezar el proceso de simulación. Limes también facilita el tiempo actual de simulación para sincronizar los dos simuladores.
- *InsertRequest*. Esta orden indica al NetSim que un nuevo mensaje tiene que ser inyectado en la red. Limes facilita toda la información necesitada: nodo origen, nodo destino, longitud del mensaje y tiempo de generación. También facilita un índice de petición, así cuando el mensaje llega a su destino, el simulador de memoria sabrá identificar el mensaje.
- *Simulate*. El simulador de memoria ordena al NetSim la simulación desde el último tiempo simulado (por el simulador de redes) hasta el tiempo actual de simulación. Este periodo de tiempo es seguro para ser simulado por el NetSim porque ya ha sido simulado por el Limes. El simulador de memoria es detenido hasta que el NetSim simula el periodo de tiempo. La figura 3 muestra el control de tiempos entre el Limes y el NetSim. Cuando el NetSim alcanza el tiempo actual de simulación del simulador de memoria, se detiene y el simulador de memoria continua su ejecución.

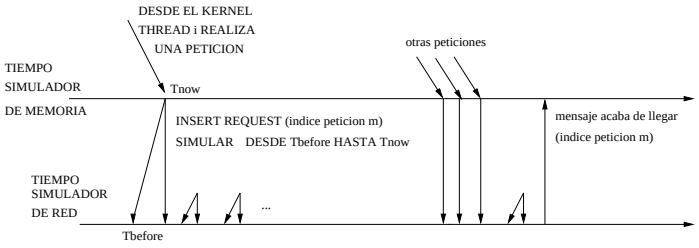


Fig. 3. Control de tiempos entre los dos simuladores

- *EndNetSim*. Esta orden avisa al simulador de redes que la simulación ha finalizado, por lo que hay que recoger las estadísticas.
  Por otra parte, el simulador de redes puede enviar las siguientes órdenes al simulador de memoria:
- *MessageArrived*. Cuando el NetSim detecta que un mensaje llega a su nodo destino, envía el índice de petición al simulador de memoria.
- *TimeSimulated*. NetSim indica al simulador de memoria que acaba de simular el periodo de tiempo solicitado.

El simulador de memoria controla los hilos de la aplicación. Cuando necesita enviar una petición a la red, el hilo involucrado es detenido hasta que todos los mensajes originados por la petición se han completado. El siguiente ejemplo muestra la interacción entre los dos simuladores para un DSM con un modelo NCC-NUMA. Se asume que se quiere leer una posición remota de memoria:

- El simulador de memoria obtiene el nodo destino, envía la orden *InsertRequest* al simulador de redes con el mensaje de tipo **Read Request** (y los apropiados parámetros) y detiene el hilo.
- El simulador de redes recibe la petición e inyecta el mensaje en el tiempo indicado.
- Limes y NetSim simulan periodos de tiempo de forma alternativa.
- Algunos ciclos después, el mensaje llega a su destino y el simulador de redes envía la orden *MessageArrived* al simulador de memoria.
- El simulador de memoria simula el comportamiento en el nodo destino, enviando de vuelta la información solicitada. Así, envía la orden *InsertRequest* al NetSim con el mensaje **Read Ack**. El hilo sigue bloqueado.
- El simulador de redes recibe la petición e inyecta el nuevo mensaje en el tiempo indicado.
- Limes y NetSim simulan periodos de tiempo de forma alternativa.
- Algunos ciclos después, el nuevo mensaje llega a su nodo destino (que es el nodo que generó el **Read Request**). El simulador de redes envía la orden *MessageArrived* al Limes.
- El simulador de memoria despierta el hilo para que continúe su ejecución.

## V. Evaluación del simulador

En esta sección presentamos, algunos resultados de evaluación de una arquitectura DSM con un modelo de memoria de tipo NCC-NUMA. En este modelo no se usan caches. En la figura 4 podemos ver el diagrama de bloques donde la memoria principal esta distribuida entre todos los nodos. Cuando un procesador lanza una petición de memoria, se calcula el nodo que tiene esa posición. Si la dirección esta localizada en la memoria local, la petición es servida en el mismo ciclo. Por el contrario, si la dirección es remota, se envía un mensaje de petición al nodo que la posee, siguiendo un protocolo *request-reply*. El procesador se detiene hasta que lleguen los reconocimientos correspondientes. En la figura 5 se muestra el protocolo *request-reply*.

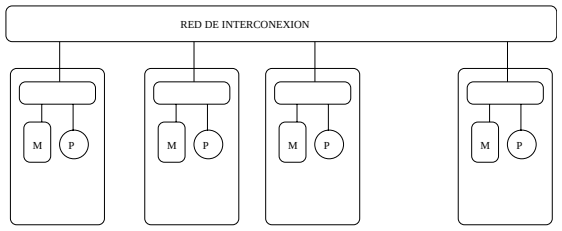


Fig. 4. Diagrama de bloques del modelo NCC-NUMA

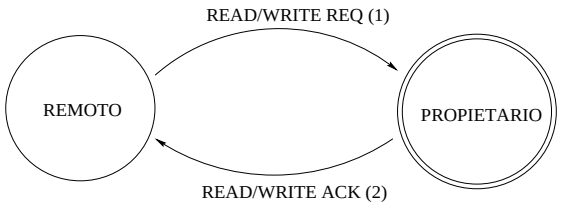


Fig. 5. Protocolo Request-reply

La distribución de los datos en memoria es crucial en este modelo. Cuanta más localidad encuentren los procesadores, mejores resultados se obtendrán. En particular, algunas aplicaciones SPLASH-2[2] son muy sensibles al patrón de distribución de los datos. Otro aspecto crítico es la asignación de procesos a procesadores. Cada proceso debe ser

asignado a un procesador de tal forma que la localidad de los datos sea óptima. En esta evaluación se han usado las mejores estrategias de distribuciones de datos y asignación de procesos.

La red de interconexión es una red de 64 nodos con una topología del tipo k-ario n-cubo con n=2 y conmutación *wormhole*. Se utilizan tres algoritmos de encaminamiento: determinista, parcialmente adaptativo y totalmente adaptativo. Los algoritmos de encaminamiento parcial y totalmente adaptativos fueron propuestos en [5]. El algoritmo parcialmente adaptativo usa dos canales virtuales por canal físico, recorre las dimensiones de la red en orden ascendente y elige cualquier canal virtual si el nodo destino en la dimensión actual no requiere atravesar el canal de borde. El algoritmo totalmente adaptativo usa tres canales virtuales por canal físico. Los dos primeros canales virtuales son elegidos como en el algoritmo parcialmente adaptativo. Además, las dimensiones pueden ser recorridas en cualquier orden, usando el tercer canal virtual. El algoritmo determinista es el propuesto en [3] para la topología k-ario n-cubo, modificándolo para usar canales bidireccionales. Usa dos canales virtuales por canal físico como el algoritmo parcialmente adaptativo.

Como aplicaciones utilizaremos dos aplicaciones de SPLASH-2: OCEAN y FFT, usando diferentes complejidades para cada una de ellas y asignando un único proceso a cada procesador.

Como modelo de memoria alternativo usaremos el modelo PRAM (Perfect RAM). En este modelo, cada petición dura un solo ciclo. El modelo PRAM lo usaremos como referencia para determinar las prestaciones de la aplicación sin las latencias debidas a la arquitectura subyacente.

Como índice de prestaciones utilizaremos el tiempo de ejecución de las aplicaciones ante los distintos algoritmos de encaminamiento propuestos. Teniendo en cuenta que la fase de inicialización no puede ser paralelizada, no la incluiremos en la evaluación. Esta aproximación es la utilizada en [2].

Diferenciaremos dos posibles contextos para las simulaciones, en un primer contexto analizamos las aplicaciones sin carga adicional en la red (esto modela a una máquina DSM en la que solamente hay una aplicación ejecutándose). En otro contexto analizaremos las aplicaciones cuando la red ya está soportando cierto nivel de carga (esta subcarga será de tipo uniforme y representará la carga generada por otras aplicaciones).

### *A. Resultados de las simulaciones sin subcarga*

La figura 6 muestra los tiempos de ejecución para la aplicación OCEAN. Como podemos ver, el algoritmo de encaminamiento parcialmente adaptativo mejora el tiempo de ejecución al menos un 40% con respecto al algoritmo determinista. Por otra parte, como era de esperar, el algoritmo de encaminamiento totalmente adaptativo obtiene mejores resultados que el determinista y el parcialmente adaptativo, reduciendo el tiempo de ejecución en un 70% y un 20% respectivamente. Sin embargo, el tiempo de ejecución alcanzado por el algoritmo de encaminamiento totalmente adaptativo está aún lejos del modelo PRAM (en un factor de 7, aproximadamente). Recordar que el modelo de memoria considerado (NCC-NUMA) es altamente ineficiente, ya que no implementa memorias cache para las direcciones remotas.

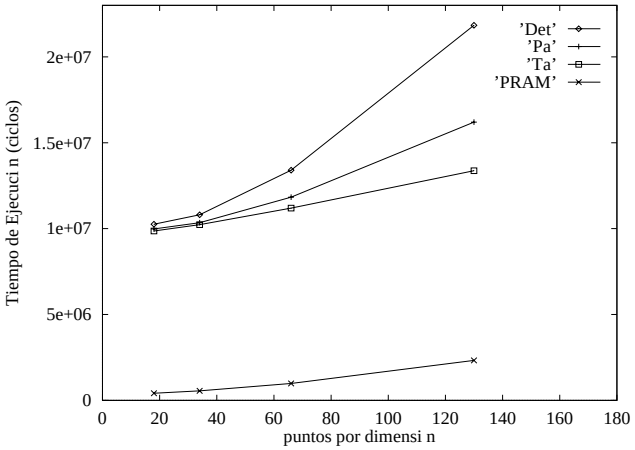


Fig. 6. Tiempos de ejecución para algunas complejidades en OCEAN

La figura 7 muestra el tiempo de ejecución de la aplicación FFT. El algoritmo de encaminamiento parcialmente adaptativo reduce el tiempo de ejecución en un 30% sobre el determinista. Aunque el totalmente adaptativo mejora ligeramente el parcialmente adaptativo, la mejora es menor que en OCEAN. Este comportamiento es debido al hecho de que la aplicación FFT no genera tanto tráfico como OCEAN, de forma que la flexibilidad ofrecida en el encaminamiento totalmente adaptativo es poco útil.

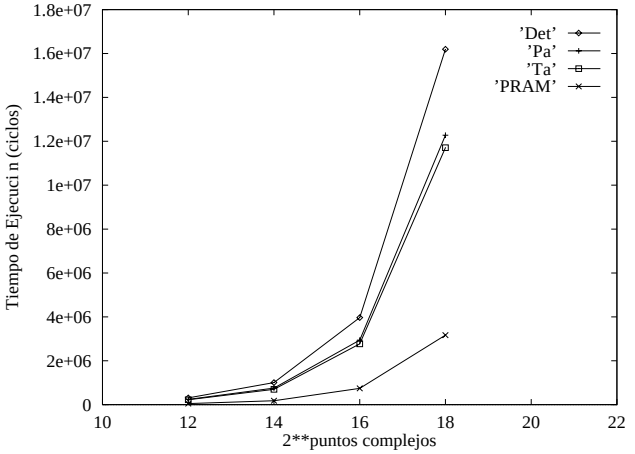


Fig. 7. Tiempos de ejecución para algunas complejidades en FFT

En la figura 8 podemos observar la productividad media para la aplicación OCEAN. Como puede verse, esta aplicación genera muy poco tráfico de red. La productividad alcanzada por el algoritmo totalmente adaptativo llega a ser de 0.06 flits/nodo/ciclo usando la mayor complejidad. En este tipo de redes, el ancho de la bisección suele ser un orden de magnitud mayor que el obtenido por OCEAN. Esto se debe al modelo de consistencia secuencial que hemos utilizado en el subsistema de memoria, de forma que la inyección de mensajes se limita a 64 en un instante de tiempo determinado.

La figura 9 muestra la latencia media para la aplicación OCEAN, corroborando los resultados obtenidos en la gráfica anterior. El análisis de productividad y latencia para FFT (no mostrado) nos lleva a resultados similares.

La mejora en prestaciones del algoritmo de encaminamiento adaptativo, con muy baja carga y con los modelos de memoria analizados, nos lleva a esperar importantes

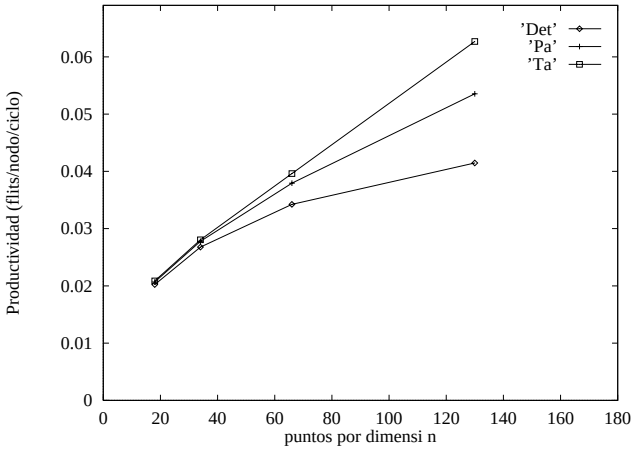


Fig. 8. Productividad OCEAN

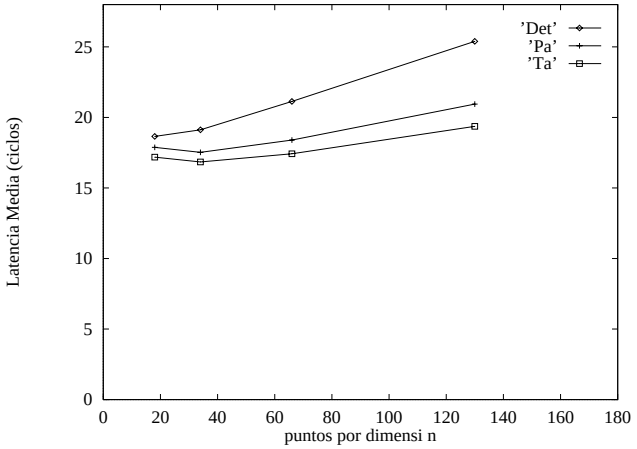


Fig. 9. Latencia OCEAN

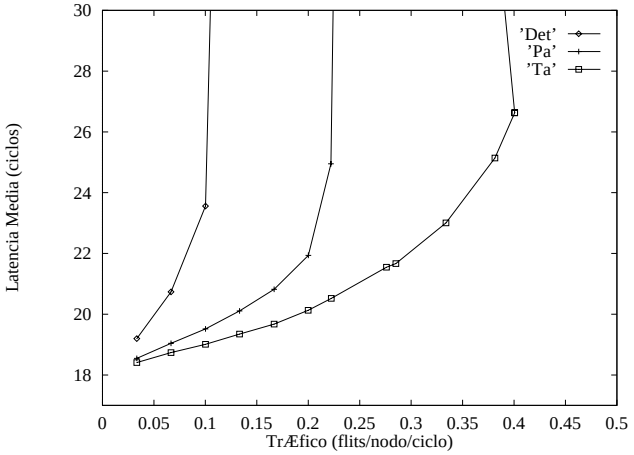


Fig. 10. Subcarga del sistema

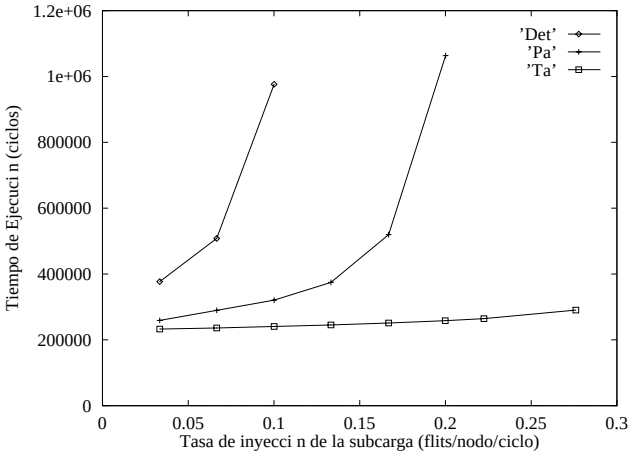


Fig. 11. Tiempos de ejecución para FFT con subcarga

mejoras en los tiempos de ejecución de aplicaciones que requieran un mayor uso de la red, así como modelos de memoria más eficientes (CC-NUMA, COMA).

### *B. Resultados de las simulaciones con subcarga uniforme*

En la figura 10 podemos observar las prestaciones obtenidas por la red cuando el tráfico procede únicamente de la subcarga. En concreto, se ha utilizado una subcarga con distribución uniforme de destinos y diferentes tasas de generación de tráfico. Los mensajes enviados son de 4 flits. En la figura observamos la saturación que provoca la subcarga en función del algoritmo de encaminamiento utilizado. En concreto, el algoritmo determinista se satura con productividad igual a 0.12 flits/nodo/ciclo, el algoritmo parcialmente adaptativo con productividad igual a 0.23 flits/nodo/ciclo, mientras que el algoritmo totalmente adaptativo se satura con una productividad de 0.4 flits/nodo/ciclo.

Utilizando esta subcarga y lanzando la ejecución de una aplicación FFT con complejidad de 2048 puntos, se obtienen los tiempos de ejecución mostrados en la figura 11. Nótese que ahora el tráfico aceptado por la red es la suma del generado por la subcarga y el originado por la aplicación. Podemos observar una apreciable diferencia entre los tres algoritmos de encaminamiento. Basándonos en el tercer punto de simulación, las mejoras del algoritmo totalmente adaptativo con respecto al determinista y parcialmente adaptativo son del 80% y del 20%, respectivamente.

## VI. Conclusiones

En este artículo hemos expuesto la necesidad de realizar una evaluación precisa basándonos en cargas reales, en lugar de utilizar cargas sintéticas o cargas dirigidas por traza. Hemos diseñado un nuevo simulador de redes dirigido por ejecución, que permite la evaluación de redes de interconexión para DSMs usando cargas de aplicaciones reales como las SPLASH-2. La herramienta también puede ser utilizada para el análisis de prestaciones del subsistema de memoria. Como ejemplo, hemos desarrollado un modelo de memoria NCC-NUMA simulando la ejecución de las aplicaciones con diferentes algoritmos de encaminamiento. Los resultados muestran que el encaminamiento adaptativo mejora las prestaciones (tiempo de ejecución) sobre el encaminamiento determinista como mínimo en un 30%.

También hemos analizado el comportamiento de estos algoritmos de encaminamiento cuando la red soporta, además de los mensajes generados por la aplicación, una carga subyacente correspondiente a una distribución uniforme de destinos. En este caso observamos que el tiempo de ejecución de las aplicaciones depende poco de la carga existente en la red cuando usamos un algoritmo de encaminamiento totalmente adaptativo, permitiendo la coexistencia de varias aplicaciones sin degradar su tiempo de ejecución.

Como trabajo futuro, pretendemos evaluar las redes de interconexión usando modelos de memoria más realistas

como CC-NUMA, COMA, Simple COMA y otros modelos de consistencia de memoria. También estamos interesados en evaluar redes irregulares de estaciones de trabajo (NOWs).

## Referencias


[1] A. Agarwal et al., "The MIT Alewife Machine: Architecture and Performance," *Proceedings of the 22nd on Computer Architecture*, June 1995.

[2] S. Cameron Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp 24-36, June 1995.

[3] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[4] H. Davis, S.R. Goldschmidt and J.Hennessy, "Multiprocessor and Tracing Using Tango," *Proceedings of the Conference on Parallel Processing*, pp 99-107, Aug. 1991.

[5] J. Duato and P. López, "Performance evaluation of adaptive routing algorithms for k-ary n-cubes," *Parallel Computer Routing and Communication*, K. Bolding and L. Snyder (ed.), Springer-Verlag, pp. 45–59, 1994.

[6] S. Fortune and J.Wyllie, "Parallelism in Random Access Machines," *Proceedings of the 10th ACM Symposium on Theory of Computing*, May 1978.

[7] S. Goldschmidt, "Simulation of Multiprocessors: Accuracy and Performance," *Ph.D. Thesis, Stanford University*, June 1993.

[8] Al Geist et al., *Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press.

[9] R.E. Kessler and J.L. Schwarzmeier, "CRAY T3D: A new dimension for Cray Research," *Compcon*, pp. 176–182, Spring 1993.

[10] J. Kuskin et al., "The Stanford FLASH Multiprocessor," *Proceedings of the 21st International Symposium on Computer Architecture*, pages 302-313, Chicago, IL, April 1994.

[11] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz and M. Lam, "The Stanford DASH multiprocessor," *IEEE Computer*, vol. 25, no. 3, pp. 63–79, March 1992.

[12] R.J. Littlefield, "Characterizing and tuning communications performance for real applications," *Proceedings of the First Intel DELTA Applications Workshop*, February 1992.

[13] E. Lusk et al., "Portable Programs for Parallel Processors," *Holt, Rinehart and Winston*, Orlando, Fl, 1987.

[14] D. Magdic, "Limes: A Multiprocessor Simulation Environment," *TCCA Newsletter*, pp 68-71, March 1997.

[15] M.P. Malumbres, J. Duato and J. Torrellas, "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors," in *Proc. of the eighth IEEE Symp. on Parallel and Distributed Processing*, pp. 186-189, October 1996.

[16] J. Protic, I. Tartalja, V. Milutinovic, "Distributed Shared Memory: Concepts and Systems", *IEEE Parallel and Distributed Technology*, Vol.5, No 1, Summer 1996.

[17] M. Snir et al., *MPI: The Complete Reference*, Cambridge, MA, MIT Press, 1996.

[18] F. Silla, M. P. Malumbres, J. Duato, D. Dai and D. K. Panda, "Impact of Adaptivity on the Behavior of Networks of Workstations under Bursty Traffic", submitted to *International Conference on Parallel Processing 1998*.

[19] R. Stets et al., "CASHMERE-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network, *Proceedings, SOSP '97*, Saint Malo, France, October 1997.

[20] C. B. Stunkel and W. K. Fuchs, "TRAPEDS: Producing Traces for Multicomputers Via Execution Driven Simulation", *Performance Review*, pp 70-78, May 1989.

[21] D. Thiebaut, J.L. Wolf, H. S. Stone, "Synthetic Traces for Trace-Driven Simulation of Cache Memories", *IEEE Transactions on Computers*, vol. 41, april 1992.