UNIVERSIDAD DE CANTABRIA

DPTO. DE ELECTRÓNICA Y COMPUTADORES.

IMPACTO DEL SUBSISTEMA DE COMUNICACIÓN EN EL RENDIMIENTO DE LOS COMPUTADORES PARALELOS: DESDE EL HARDWARE HASTA LAS APLICACIONES.

Presentada por:

Valentin Puente Varona

Dirigida por:

Ramón Beivide Palacio.

SANTANDER, OCTUBRE DE 1999

Apéndice A

Herramientas e Infraestructura de Simulación.

En este Apéndice se describe la infraestructura de simulación empleada a lo largo de la tesis presentada en esta memoria. La herramienta central es un simulador de redes de interconexión directas, altamente flexible, que permite modelar una amplia variedad de encaminadores de modo muy preciso. En este sentido, el objetivo perseguido con este simulador ha sido, fundamentalmente, el disponer de una herramienta que trate, de una forma homogénea, la evaluación del rendimiento de encaminadores hardware sobre redes realistas, permitiendo evitar los altos requerimientos de tiempo y recursos que presentan las simulaciones hardware, sin por ello perder precisión con respecto a los resultados obtenidos. Además, se ha desarrollado un entorno más amplio, sobre su base, que permite la evaluación del rendimiento de la red en términos de cargas reales.

A.1 Introducción.

En este capítulo se describirán de forma concisa las características básicas del entorno de trabajo y herramientas que han sido empleadas a lo largo de esta tesis. Dada la complejidad de los sistemas evaluados es necesario plantearse una metodología de evaluación en base a simulación. Si bien existe un amplio abanico de técnicas analíticas que pueden ser aplicadas en este tipo de sistemas, la precisión alcanzada puede alejase considerablemente de la realidad.

Como base del entorno de trabajo se ha empleado un simulador de redes de interconexión desarrollado en el seno del Grupo de Arquitectura y Tecnología de Computadores(ATC) de la Universidad de Cantabria [100]. Este simulador posee una elevada precisión siendo a la vez altamente modular y extensible. Su característica fundamental es que logra aproximar considerablemente los resultados obtenidos a los que suministran los simuladores *hardware,* pero con un coste computacional notablemente inferior. La idea seguida en su desarrollo rompe con otro tipo de herramientas de idéntico cometido al establecer una interface de uso simple pero extraordinariamente potente.

Tomando como base esta herramienta se ha pretendido ir más allá, contemplando la posibilidad de evaluar el rendimiento de cada propuesta para la red de interconexión en base a cargas reales. Esta tarea ha sido abordada desde dos puntos de vista. En primer lugar, se ha logrado integrar la red evaluada por la herramienta dentro de un simulador conducido por ejecución que emula el comportamiento de una arquitectura ccNUMA y por otro lado, se ha desarrollado una metodología de evaluación para este tipo de arquitecturas en base a trazas. En conjunto, estos simuladores, junto con las herramientas de diseño y simulación *hardware,* han conformado el marco para la elaboración de todo el trabajo presentado.

En cuanto al resto del apéndice, cabe señalar que en la Sección A.2 se indicarán las características mas relevantes del simulador; en la Sección A.3 se introducirán la estructura y modo de funcionamiento del simulador conducido por ejecución y en la Sección A.4 se propondrá y evaluará una nueva metodología de evaluación de redes de interconexión para sistemas ccNUMA con procesadores ILP basada en trazas.

A.2 Simulador de Redes de Interconexión.

En la evaluación de prestaciones de las redes de interconexión, las herramientas analíticas y las técnicas de simulación frecuentemente se complementan. La aplicación de las herramientas analíticas está condicionada por la complejidad del sistema que se está modelando. A veces, para que el modelo sea tratable analíticamente, tienen que hacerse una serie de hipótesis poco

realistas sobre el sistema y/o su funcionamiento que pueden comprometer seriamente la credibilidad del modelo resultante. En estos casos, es imprescindible evaluar los sistemas a través de técnicas de simulación. Los siguientes puntos, propios del uso de la simulación, deben ser considerados muy cuidadosamente:

1. La ejecución de una simulación puede ser muy costosa, en términos de tiempo de computación, por lo que el modelo debe ser lo más simple posible sin comprometer la fidelidad de las características esenciales del sistema modelado.
2. Las simplificaciones hechas en la construcción de un modelo han de ser tenidas en cuenta a la hora de interpretar los resultados.
3. Al comienzo de la simulación se funciona en régimen transitorio, y se debe evitar realizar medidas estadísticas mientras persista este régimen.

En los últimos años, con la proliferación de las técnicas de simulación [11][12], resulta necesario disponer de una herramienta que balancee eficientemente la relación existente entre la fiabilidad de las medidas obtenidas con el tiempo empleado en obtener las mismas. Las herramientas de descripción hardware, como VHDL, permiten modelar sistemas con un bajo nivel de abstracción, por lo que son muy fiables en cuanto a la precisión de sus resultados. Sin embargo, los recursos y el tiempo empleado limitan fuertemente su uso. La solución a este problema consiste en realizar una descripción del sistema bajo estudio en un lenguaje de alto nivel, extrayendo las características más relevantes y desechando aquellas otras que compliquen el modelo innecesariamente. El nivel de abstracción, en este caso, es mucho más alto y consecuentemente los resultados son menos precisos [98]. A este respecto, cabe citar que un gran número de propuestas sobre encaminadores de mensajes, se ven acompañados de datos obtenidos a partir de modelos que son, a nuestro juicio, demasiado simples como para reflejar fielmente el comportamiento del encaminador [20]. Por este motivo, en nuestro grupo se ha diseñado y construido un simulador en un lenguaje de alto nivel que describe un bajo nivel de abstracción sobre el *hardware* que modela. Este simulador se ha denominado *SICOSYS* (SImulator of COmunication SYStems) [100].

El simulador proporciona una solución al problema genérico, esto es, podemos emplear el mismo entorno de simulación cuando deseemos experimentar con distintas propuestas arquitectónicas. En este sentido se han identificado los principales componentes hardware que forman parte del encaminador: memorias, unidades de encaminamiento, multiplexores y conmutadores. Como características fundamentales del sistema a modelar, en principio, son las dos siguientes:

1. Redes de interconexión directas en 2 y 3 dimensiones.
2. Encaminadores diseñados con tecnología síncrona y con una interfaz de comunicación asíncrona.

A.2.1 Características Básicas de SICOSYS.

Con objeto de disponer de una arquitectura software para el simulador que sea fácilmente comprensible, extensible y reutilizable, se ha realizado un diseño de la aplicación orientado a objetos, con una implementación final en lenguaje C++.

La implementación del simulador se basa en una nueva tecnología software que está íntimamente ligada a un diseño orientado a objetos, como son los patrones de diseño [56]. En concreto, han sido necesarias aproximadamente 110 clases, distribuidas en casi 50.000 líneas de código. La implementación se ha realizado teniendo en cuenta la portabilidad, por lo que no se utiliza ninguna característica propia del entorno de desarrollo que entorpezca la posibilidad de utilizar el simulador en diferentes plataformas.

El simulador es altamente configurable, en el sentido de que podemos especificar exactamente el tipo de experimento que deseamos realizar a través de unos ficheros de descripción. Estos ficheros de entrada utilizan el lenguaje SGML[6], para describir con precisión el experimento a través de 3 aspectos diferentes:

1. *Arquitectura hardware:* un encaminador se compone de una serie de elementos, tales como memorias, conmutadores, etc. conectados entre sí. Se trata de describir esta estructura, así como de fijar las características funcionales y temporales de cada elemento.
2. *Red de interconexión*: se especifica el tipo de red utilizada (malla, toro, …), así como sus dimensiones y retraso de los cables entre los encaminadores.
3. *Parámetros de la simulación*. Cuestiones tales como: tiempo que dura el experimento, tipo de tráfico, carga aplicada, longitud de los mensajes, etc.

Por otro lado, la estructura software del simulador puede subdividirse en las siguientes secciones:

- **Constructores *SGML***: se encargan de interpretar la descripción *SGML* de los ficheros de entrada al simulador y generar los objetos requeridos para configurar la simulación. Existen 3 constructores, cada uno de los cuales se encarga de definir una parte diferente del experimento:

•Encaminador a ser utilizado.

•Tipo de red de interconexión.

•Parámetros de la simulación.

- **Componentes**: representan el hardware que vamos a simular: la red y el encaminador con sus buffers, unidades de decisión, conexiones, conmutadores, multiplexores, etc. Se modela tanto el comportamiento, como las características temporales de cada componente. De esta forma, se establece una relación directa entre una estructura *hardware* concreta y el modelo que genera nuestro simulador. El *software* desarrollado es muy flexible a la hora de crear nuevos componentes. Los constructores *SGML* simplemente generan una jerarquía de componentes relacionados entre sí.
- **Patrones de tráfico**: los modelos adoptados son en principio de tipo sintético (uniforme, matriz transpuesta, bit reversal, perfect shuffle, hot-spot, etc...). El tráfico generado puede ser de tipo bimodal, esto es, los mensajes pueden tener dos longitudes diferentes.
- **Simulador**: una vez construida la estructura que modela el hardware, se añaden las características de nivel general que van a determinar el experimento: patrón de tráfico, carga aplicada, longitud de los mensajes, etc. Se almacenan los valores de los parámetros a medir más relevantes durante la ejecución del experimento, y cada determinado número de ciclos genera información sobre el estado de la simulación, que puede ser captado por un interfaz (ya sea en modo texto o en modo gráfico) para filtrar esa información y poder extraer la que más interese.

La principal aportación de este esquema es que con un único simulador podemos configurar experimentos completamente diferentes sin más que operar sobre los ficheros de entrada. No es necesaria ninguna recompilación del código fuente. Aunque esto supone una pérdida de prestaciones en la fase de inicialización del experimento a realizar, se obtiene como compensación el disponer de una forma homogénea de realizar los experimentos, así como una gran facilidad en el uso de la herramienta. Cada experimento generará como resultado un conjunto de medidas, tales como la latencia máxima y media de los mensajes, la carga aplicada y la aceptada, etc. Además, es posible solicitar al simulador que realice medidas específicas a nivel global (en toda la red) o concreto (en uno o varios encaminadores) de cuestiones tales como la frecuencia de recepción de mensajes, ocupación de las memorias, etc. A modo de ejemplo, en la Tabla A-1 se muestra la descripcion SGML de un encaminador determinista para redes toroidales con mecanismo de evitacion de interbloqueo basado en la burbuja asi como el fichero de red y simulacion.

```
<!--*********************************************************************-->
<!-- Descripcion SGML de un encaminador determinista para redes toroidales -->
<!-- con mecanismo de evitacion de interbloqueo basado en la burbuja-->
<!--*********************************************************************>
<Router id="DOR2D-BU" inputs=4 outputs=4 bufferSize=64 bufferControl=CT routingControl="DOR-BU">
   <Injector id="INJ">
   <Consumer id="CONS">

   <Buffer id="BUF1" type="X+" dataDelay=2>
   <Buffer id="BUF2" type="X-" dataDelay=2>
   <Buffer id="BUF3" type="Y+" dataDelay=2>
   <Buffer id="BUF4" type="Y-" dataDelay=2>
   <Buffer id="BUF5" type="Node" dataDelay=2>

   <Routing id="RTG1" type="X+"   headerDelay=1 dataDelay=0>
   <Routing id="RTG2" type="X-"   headerDelay=1 dataDelay=0>
   <Routing id="RTG3" type="Y+"   headerDelay=1 dataDelay=0>
   <Routing id="RTG4" type="Y-"   headerDelay=1 dataDelay=0>
   <Routing id="RTG5" type="Node" headerDelay=1 dataDelay=0>

   <Crossbar id="CROSSBAR" inputs="5" outputs="5" type="CT" headerDelay=2 dataDelay=1>
      <Input  id=1  type="X+">
      <Input  id=2  type="X-">
      <Input  id=3  type="Y+">
      <Input  id=4  type="Y-">
      <Input  id=5  type="Node">

      <Output id=1  type="X+">
      <Output id=2  type="X-">
      <Output id=3  type="Y+">
      <Output id=4  type="Y-">
      <Output id=5  type="Node">
   </Crossbar>

   <Connection id="C01" source="INJ" destiny="BUF5">
   <Connection id="C02" source="CROSSBAR.5" destiny="CONS">

   <Connection id="C03" source="BUF1" destiny="RTG1">
   <Connection id="C04" source="BUF2" destiny="RTG2">
   <Connection id="C05" source="BUF3" destiny="RTG3">
   <Connection id="C06" source="BUF4" destiny="RTG4">
   <Connection id="C07" source="BUF5" destiny="RTG5">

   <Connection id="C08" source="RTG1" destiny="CROSSBAR.1">
   <Connection id="C09" source="RTG2" destiny="CROSSBAR.2">
   <Connection id="C10" source="RTG3" destiny="CROSSBAR.3">
   <Connection id="C11" source="RTG4" destiny="CROSSBAR.4">
   <Connection id="C12" source="RTG5" destiny="CROSSBAR.5">

   <Input id="1" type="X+"   wrapper="BUF1">
   <Input id="2" type="X-"   wrapper="BUF2">
   <Input id="3" type="Y+"   wrapper="BUF3">
   <Input id="4" type="Y-"   wrapper="BUF4">

   <Output id="1" type="X+"   wrapper="CROSSBAR.1">
   <Output id="2" type="X-"   wrapper="CROSSBAR.2">
   <Output id="3" type="Y+"   wrapper="CROSSBAR.3">
   <Output id="4" type="Y-"   wrapper="CROSSBAR.4">

</Router>
<!--************************************************-->
<!-- Fichero de red para el encaminador anterior-->
<!************************************************-->
<TorusNetwork id="T88-DOR2D-BU" sizeX=8 sizeY=8 router="DOR2D-BU" delay=0>

<!--************************************************-->
<!-- Fichero de simulacion para la red anterior -->
<!-- ( Tráfico nbasado en la traza de una aplicacion)-->
<!************************************************-->
<Simulation id="FFT_BIG_T88-DOR2D-BU">
   <Network            id="T88-DOR2D-BU">
   <SimulationCycles   id=100000>
   <TrafficPattern     id="MODAL" type="TRACE" traceFile="../DB/fft.64.big.hpc.8.32.1a2" flitsz=4 static=1
compress=1  clock=2.02>
   <MessageLength      id=1>
</Simulation>
```

Tabla A-1. Ficheros SGML de simulación para encaminador determinista burbuja.

La determinación del momento en el que se debe finalizar una simulación es uno de los aspectos más importantes para que los resultados finales puedan resultar fiables. En otras palabras, es absolutamente necesario distinguir el período transitorio de la simulación para realizar medidas seguras en el estado estacionario. De esta forma, es posible indicar cuánto va a durar la simulación ya sea de forma automática (se delega la responsabilidad en el propio simulador) o manual (especificamos un tiempo de simulación, o bien un mínimo de mensajes a recibir).

En este sentido, se ha desarrollado una aplicación visual, implementada sobre la librería gráfica multiplataforma Qt [107]. Desde esta herramienta es posible configurar totalmente el experimento que deseamos llevar a cabo a través del cuadro de diálogo mostrado en la Figura A-1. Además, es posible observar la evolución de la latencia media de la red en función del tiempo (Figura A-2), con lo que podemos determinar, visualmente, cuándo hemos alcanzado el estado estacionario para poder detener el experimento en un lugar en el que estamos seguros de que las medidas realizadas son correctas. También es posible visualizar de forma dinámica cual es la ocupación en cada instante de los buffers, lo que nos permite observar el grado de ocupación medio de la red así como posibles zonas en las que haya una sobrecarga de tráfico.

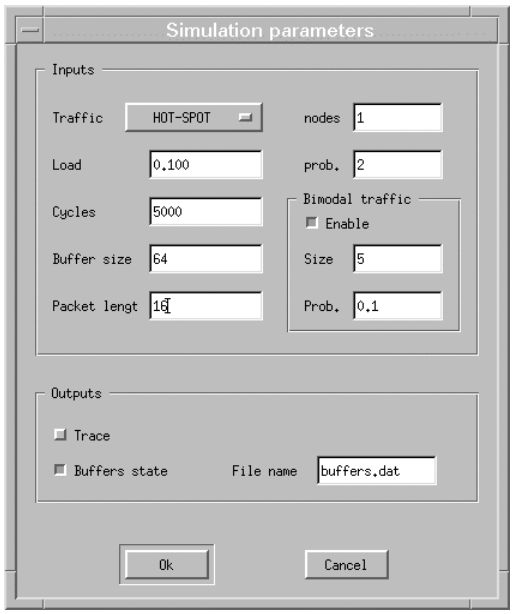

Figura A-1. Parámetros de entrada del experimento.

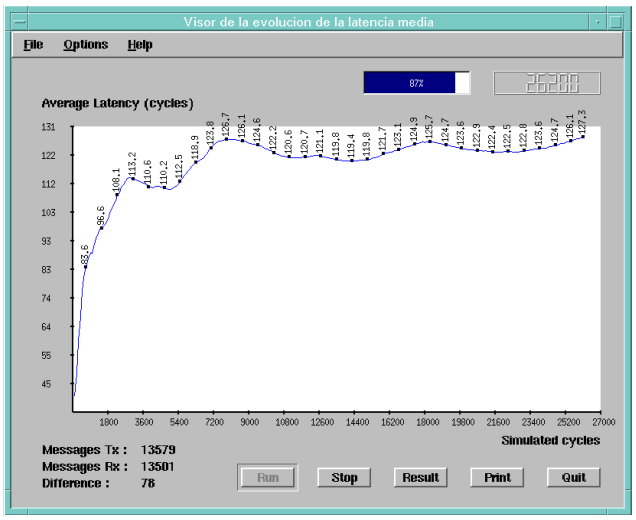

Figura A-2. Representación de la Evolución Temporal de la Latencia Media.

A.2.2 Análisis de Rendimiento y Comparación con otras Alternativas de Simulación.

A continuación pasaremos a analizar alguno de los resultados obtenidos con el simulador presentado en este trabajo, frente a dos alternativas claramente diferenciadas.

En primer lugar, se comparan los resultados obtenidos por el simulador frente a simulaciones VHDL. Como ejemplo prueba, se ha seleccionado un encaminador adaptativo sobre redes toroidales 2D descrito en VHDL y del que, en el simulador SICOSYS, se han contemplado las peculiaridades más importantes de su implementación *hardware*.

En segundo lugar, se comparan los resultados del simulador detallado con un simulador funcional de complejidad mucho más reducida. El encaminador seleccionado en esta segunda prueba es determinista, siendo más sencillo que el utilizado en la primera prueba. Este encaminador ha sido implementado en VHDL, quedando los detalles de dicha implementación contemplados en el simulador SICOSYS. Sin embargo, dada la sencillez del simulador funcional, en éste sólo se han considerado aspectos referentes a la funcionalidad del encaminador, no siendo posible tener en cuenta detalles dependientes de la implementación *hardware*.

A.2.2.1 Simulador SICOSYS frente a Simulador VHDL.

La estructura del router seleccionada en esta fase de la comparación es la representada en la Figura A-3. Se trata de un encaminador completamente adaptativo *Virtual Cut-Through*, con un mecanismo de evitación de bloqueos basado en restricción de inyección. En el Capítulo 3 se puede encontrar una descripción detallada del mismo.

A partir de las especificaciones originales del encaminador se ha hecho una descripción funcional en VHDL y tras sucesivos procesos de depuración se ha alcanzado una descripción totalmente sintetizable por la herramienta EDA de la forma en que se describe en la Sección 2.2. La librería tecnologíca empleada es *ECPD07* (0.7μm). Una vez alcanzada la descripción sintetizable y de acuerdo con las especificaciones impuestas originalmente, el siguiente paso es evaluar las prestaciones finales de la implementación.

En este punto es necesario evaluar el rendimiento del encaminador en el contexto de una red completa. Como configuración seleccionada para esta red, se ha optado por un toro 2D con 64 nodos. Afrontar este tipo de pruebas mediante un simulador VHDL implica un gran coste en cuanto al tiempo de simulación, dados los requerimientos de cálculo que requiere este problema.

Los simuladores VHDL utilizados han sido *Vhdlsim* y *Leapfrog*. Estos simuladores corresponden respectivamente a las herramientas *Synopsys1997*.08 y *CadenceDFWII97A*. Para dar una idea inicial de rendimiento, cabe citar que los tiempos de simulación para la red comentada anteriormente se sitúan para 20.000 ciclos de simulación en aproximadamente en 10 h. para *Vhdlsim* y 2.5 h. para *Leapfrog* sobre una SunULTRA2 con 128Mb de memoria RAM. Si bien es cierto que estos valores oscilan levemente en función de la carga aplicada a la red y el tipo de tráfico considerado, las diferencias entre ambos simuladores se mantienen. A tenor de estos resultados se ha optado por utilizar el simulador VHDL de *Cadence (Leapfrog)* en esta comparativa.

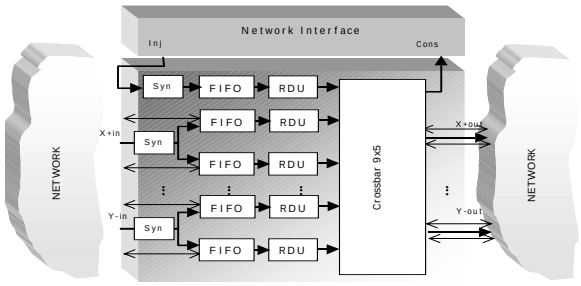

Figura A-3. Encaminador Prueba Utilizado en la Comparación SICOSYS Frente a Simulación *Vhdl*.

Como muestra de los resultados obtenidos, en la Figura A-4 se representa el número de ciclos de red simulados por segundo de CPU, para una carga sintética, con patrón de destinos uniforme, en un toro 8x8, y en la y Figura A-5 el error relativo en la latencia promedio entre ambos simuladores.

Como se puede observar, hay una notable diferencia entre el rendimiento alcanzado por los dos imuladores, llegando a ser SICOSYS, 45 veces más rápido que el simulador VHDL. Sin embargo, el error promedio cometido en la latencia por el simulador SICOSYS es reducido,

manteniéndose siempre por debajo del 4%. En el caso del *throughput* el error es es inferior. Los mismos resultados se repiten con otros encaminadores, con diferentes patrones de tráfico y configuraciones de red. A la vista de estos resultados, cabe decir que el simulador SICOSYS es una alternativa perfectamente viable para probar el rendimiento de las implementaciones VHDL de los encaminadores bajo un amplio rango de condiciones, sin perdida de precisión. Esto nos permite garantizar que los resultados mostrados a lo largo de todo el trabajo son altamente fiables.

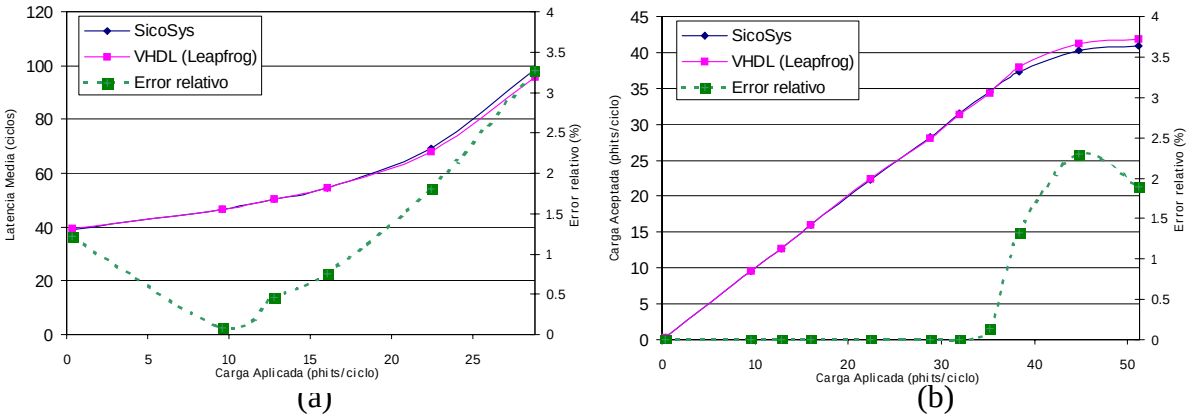

Figura A-4. (a)Latencia Promedio de SICOSYS frente a *Leapfrog* y error relativo, (b) *Throughput.*.

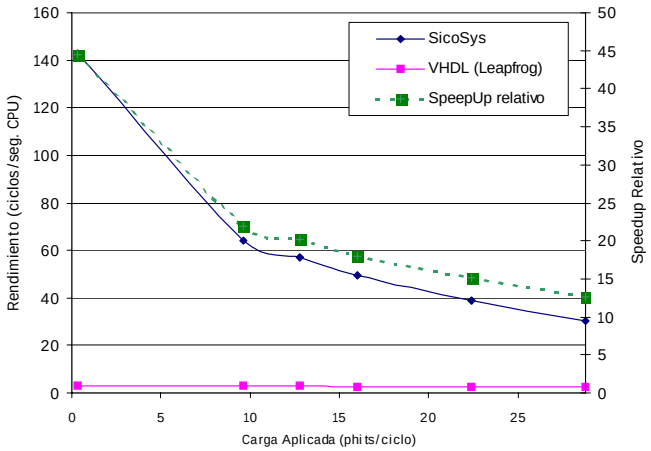

Figura A-5. Rendimiento de SICOSYS frente a *Leapfrog* y ganancia relativa.

A.2.2.2 SICOSYS frente a Simulador Funcional.

En este punto analizaremos el rendimiento y precisión alcanzada por nuestro simulador frente a un simulador funcional convencional. Para efectuar esta prueba se ha partido de la implementación VHDL del encaminador que aparece en la Figura A-6. Se trata de un encaminador determinista con mecanismo de evitación de deadlock basado en la burbuja. De la misma que en el empleado en el punto anterior, en el Capítulo 3 se puede encontrar una descripción pormenorizada del mismo. A partir de esta implementación se ha generado una descripción detallada para

el simulador SICOSYS quedando los resultados obtenidos válidados con respecto a los proporcionados por la simulación VHDL. Paralelamente, se ha desarrollado un simulador funcional específicamente para este encaminador. Este simulador se ha implementado en ADA95 y su tamaño se sitúa ligeramente por encima de las 500 líneas de código. Obviamente, en este simulador no es posible plasmar todas las características de la implementación hardware, únicamente se tienen en cuenta las características más relevantes del mismo.

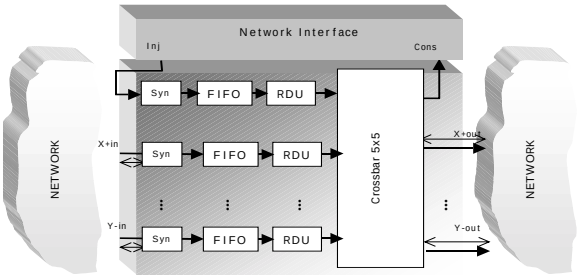

Figura A-6. Encaminador prueba utilizado en la comparación SICOSYS frente a simulador funcional.

Como muestra de la precisión alcanzada por las dos alternativas en la Figura A-7 y Figura A-8, se representan las medidas de latencia media de los mensajes en función de la carga aplicada para una red toroidal 2D de 64 nodos, construida a partir del encaminador prueba que se muestra en la Figura A-6. El patrón de tráfico utilizado es uniforme.

Figura A-7. (a)Latencia promedio de SICOSYS frente a simulador funcional y error relativo, (b) *Throughput*.

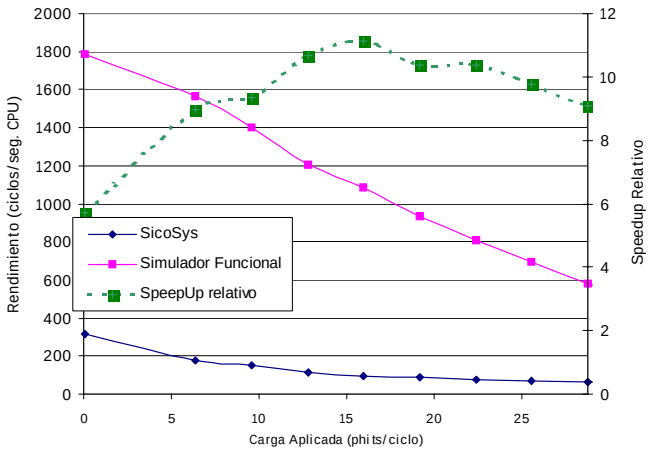

Figura A-8. Rendimiento de SICOSYS frente a simulador funcional y ganancia relativa.

A la vista de estos datos cabe resaltar la gran discrepancia entre los resultados aportados por ambos simuladores. Estas diferencias se acentúan a medida que la red se sitúa en una zona de carga más elevada. Básicamente, en esta zona de carga los detalles del propio encaminador pasan a jugar un papel realmente importante, siendo determinantes en el rendimiento final. El obviar estos detalles da lugar a errores de hasta el 30% en la latencia media observada para este caso, si bien esta diferencia tiende a acentuarse en otro tipo de patrones de tráfico o encaminadores más complejos. El error en *throughput* no es tan elevado en términos absolutos pero esas diferencias pueden tener un impacto mayor a la hora de valorar el rendimiento del sistema.

Como se puede comprobar, el tomar en consideración todos los detalles de cada encaminador hace que el rendimiento de SICOSYS sea hasta 10 veces inferior al del simulador funcional. Sin embargo, los requerimientos de tiempo son perfectamente asumibles y dado el error cometido por el simulador funcional, inevitables si deseamos obtener medidas precisas.

A.3 Cargas de Trabajo Reales: Simulador Conducido por Ejecución.

A continuación se introducirán las características más relevantes del simulador conducido por ejecución que ha sido empleado a lo largo del trabajo. Este simulador toma como punto de partida, por un lado, el simulador conducido por ejecución RSIM[95] y por otro, SICOSYS. La idea final que se ha perseguido en el desarrollo de este simulador es aunar la precisión ofrecida por el modo que tiene RSIM de simular la arquitectura superior de la máquina y la gran fiabilidad que exhibe SICOSYS a la hora de emular el comportamiento de la red de interconexión. El unir ambos simuladores permitirá alcanzar una aproximación muy buena a lo que ocurriría en arqui-

tecturas reales empleando un modelo para la red que se aproxima considerablemente al comportamiento mostrado por simuladores *hardware*.

A.3.1 Características Básicas de RSIM.

RSIM es un simulador conducido por ejecución, diseñado fundamentalmente para estudiar el comportamiento de arquitecturas de memoria compartida, escalables, con procesadores que tienen características muy evolucionadas. Comparado con otros simuladores disponibles públicamente de similares características, la principal ventaja de RSIM es que soporta procesadores que explotan de forma agresiva el paralelismo a nivel de instrucción (ILP). Esta característica permite emplear procesadores para el sistema bastante representativos y próximos a los empleados actualmente en diversos sistemas de este tipo. Otros simuladores asumen modelos para los procesadores emulados mucho más sencillos, y pueden dar lugar a imprecisiones elevadas a la hora de estudiar la interacción de este tipo de arquitecturas con la red de interconexión. El coste del incremento de precisión y detalle que muestra RSIM es que se convierte en un simulador bastante más lento que otros simuladores más simples [47].

RSIM permite definir una cantidad considerable de parámetros del sistema a emular, lo que facilita su uso en configuraciones, tanto uniprocesador como multiprocesador. Las características básicas que incorpora son:

- Características del Procesador.
    - •Múltiples vías de ejecución.
    - •Ejecución dinámica o fuera de orden.
    - •Renombrado de registros.
    - •Predicción de saltos estática o dinámica.
    - •Accesos a memoria no bloqueantes.
    - •Ejecución especulativa.
    - •Modelos de consistencia simples y optimizados.
- Características de la Jerarquía de Memoria.
    - •Dos niveles de cache.
    - •Cache de primer nivel multipuerto y segmentada. Soporte para caches de segundo nivel segmentadas.
    - •Soporte para múltiples peticiones pendientes.

- Memoria entrelazada.
- Prefech por software no bloqueante.

- Características Multiprocesador
  - Arquitectura cc-NUMA de memoria compartida con protocolo de coherencia en cache basado en directorio.
  - Soporte para MSI o MESI como protocolos de coherencia
  - Soporte para consistencia secuencial, consistencia por procesador y consistencia relajada.
  - Red de interconexión tipo malla con control de flujo Wormhole y encaminamiento determinista.

La arquitectura de la jerarquía de memoria del sistema simulado es muy similar a la empleada en el DASH [85], con la salvedad de que cada nodo del sistema solamente tiene soporte para un único procesador. En la Figura A-9 se puede apreciar un esquema del sistema de memoria del simulador.

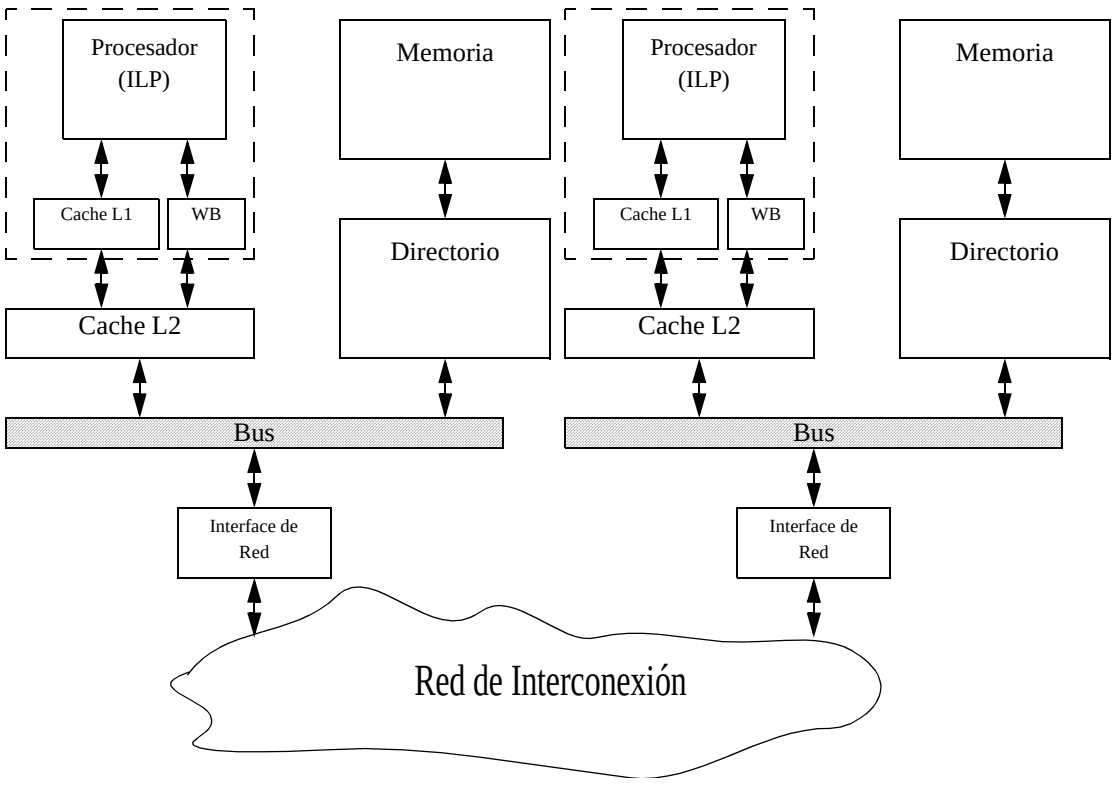

Figura A-9. Arquitectura del sistema de emulado por RSIM.

La arquitectura del procesador del sistema es muy próxima a la del MIPS R10000[91] y su estructura básica se muestra en la Figura A-10.

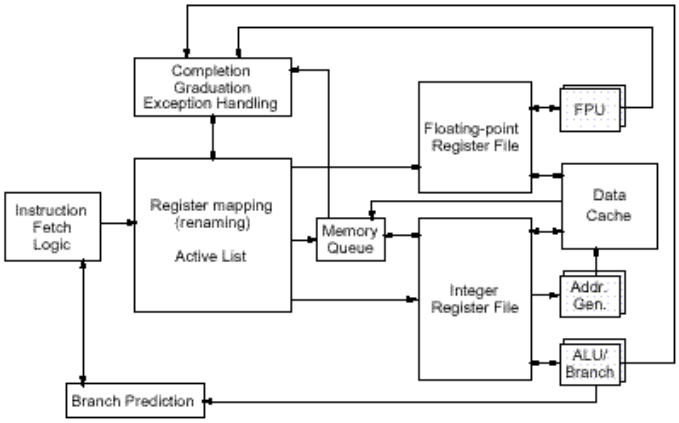

Figura A-10. Estructura del procesador emulado. Tomado de [95].

Internamente, RSIM es un simulador, en su mayor parte, conducido por eventos. El manejo de eventos está basado en la librería básica YACSIM [70]. De esta forma, la mayor parte de los sistemas que constituyen el sistema emulado son activados únicamente cuando hay algún evento asociado pendiente. Por otro lado, el procesador, dado que en la mayoría de los ciclos de la ejecución va a poseer elementos activos, está conducido por tiempo. El subsistema de memoria está escrito en C y el procesador en C++. El diseño es bastante modular y permite la modificación e incorporación de nuevos modulos sin un esfuerzo muy alto. En total el número de líneas de código se sitúa en torno a 51.000.

En cuanto a las aplicaciones que puede manejar RSIM, se puede decir que es capaz de manejar prácticamente cualquier tipo de aplicación compilada y enlazada para procesadores SPARC empleando el compilador propietario de Sun. El soporte para desarrollar aplicaciones paralelas se basa en una librería especifica que incorpora las llamadas de SystemV más empleadas usualmente para este cometido. Además, se incorpora una implementación de las macros PARCMACS[8], basadas en las llamadas básicas de SystemV. La distribución de las variables compartidas a lo largo de todo el sistema ha de ser realizada explícitamente en el código de la aplicación empleado funciones proporcionadas por la librería de desarrollo del simulador.

En principio, RSIM interpreta directamente los ejecutables de las aplicaciones compiladas sobre un sistema como el descrito anteriormente. El entorno incorpora herramientas de extracción de estadísticas de la mayor parte de los componentes del sistema, lo que permite estudiar el susb-

sistema de interés para cualquier aplicación. Además contempla la extracción de detalles, e información mucho más amplia, permitiendo, incluso, la extracción de trazas de ejecución en cualquier parte del sistema.

A.3.2 NETSIM frente a SICOSYS.

Como se ha introducido previamente, RSIM únicamente contempla como red de interconexión en el sistema emulado, una malla con control de flujo *wormhole* y encaminamiento determinista. Esta red está desarrollada sobre la base del entorno NETSIM [69]. Esta librería está pensada específicamente para el desarrollo de redes de interconexión. Sobre la configuración base, se opta por emplear dos redes, físicamente aisladas, de peticiones y datos como mecanismo de evitación de *deadlock* entre las distintas clases de tráfico [86]. Este mecanismo se ha empleado en sistemas como el *Stanford DASH*[85]. Si bien esta red es configurable en un número reducido de parámetros como anchura de los canales, retraso de los encaminadores y tamaño de los *buffers* de transito, la solución no es válida a la hora de estudiar los sistemas analizados en este trabajo.

En primera aproximación se optó por desarrollar sobre NETSIM alguna de las redes y encaminadores empleados en el trabajo. Originalmente, desarrollado de forma independiente a RSIM, el sistema ofrece muchas posibilidades a la hora de implementar encaminadores o redes específicas. Como el resto de la jerarquía de memoria de RSIM está a su vez basado en la librería básica de eventos YACSIM. En principio, NETSIM permite la utilización de tres elementos básicos: buffers, multiplexores y demultiplexores. Mediante estos tres elementos se ha de construir la red a considerar. La red se describe a través de un código C en el que se indican la estructura del encaminador con sus enlaces internos y el algoritmo de encaminamiento (característica de los demultiplexores). Además, se ha de especificar como son los enlaces de la red. La malla original de RSIM está implementada siguiendo esta filosofía. A la hora de evaluar este planteamiento para medir el rendimiento de diversas redes de interconexión con cargas reales se optó por emplearla, seleccionando para ello dos encaminadores sencillos cuya diferencias con los modulos básicos ofrecidos por NETSIM no fueran demasiado acusados. En esta línea, se consideró oportuno implementar el soporte de redes toroidales bidimensionales empleando dos tipos de encaminadores. Por un lado, un encaminador determinista con mecanismo de evitación de deadlock basado en la burbuja y otro basado en canales virtuales (Ver Sección 3.4.2).

El paso siguiente a la implementación de estas dos redes fue comparar los resultados ofrecidos por esas implementaciones frente a los alcanzados por descripciones de los mismos encaminadores sobre SICOSYS. En este análisis se emplearon exclusivamente cargas de tipo sintético.

Es posible realizar esta comparativa con las implementaciones basadas en NETSIM empleando un modulo de inyección y consumo basado en YACSIM. De esta forma, junto con las librerías básicas del entorno, es posible ejecutar simulaciones de la red de forma aislada al resto del sistema. En principio, consideramos más oportuno realizar la comparativa en términos de carga sintética que no de cargas reales ya que nos permite establecer más fácilmente las diferencias entre ambos simuladores.

Por ejemplo, en la Figura A-11 se muestran los resultados de latencia promedio alcanzados para un tráfico de tipo uniforme, sobre un toro 8x8, por cada uno de los simuladores para el caso del encaminador determinista basado en la burbuja y en la Figura A-12, para el caso del encaminador determinista basado en los canales virtuales. Como se puede apreciar, la discrepancia entre los dos simuladores llega a superar el 50% en algunos casos. Claramente, estos resultados muestran que esta primera alternativa es inviable ya que, para dos encaminadores sencillos, el modo en que se realiza la simulación influye notablemente en el resultado. Es indudable, como se mostraba en el apartado anterior, que la simplicidad de los simuladores funcionales convencionales (como NETSIM) frente a SICOSYS, a la hora de reflejar los resultados alcanzados por las descripciones hardware de los encaminadores, son considerables.

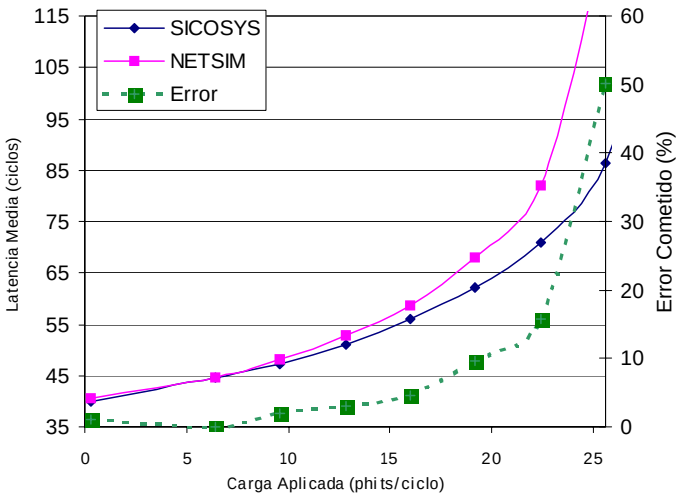

Figura A-11. Comparativa entre NETSIM y SICOSYS con un encaminador determinista basado en burbuja sobre un toro 8x8 con tráfico uniforme.

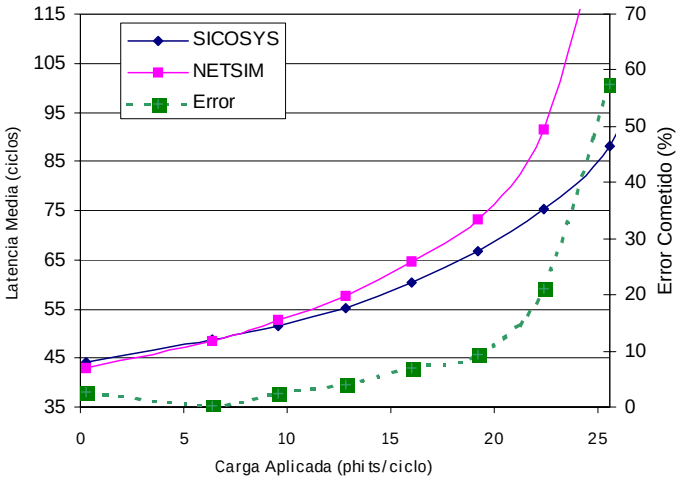

Figura A-12. Comparativa entre NETSIM y SICOSYS con un encaminador determinista basado en canales virtuales sobre un toro 8x8 con tráfico uniforme.

Estos resultados, por si solos, indican claramente que esta alternativa no es viable a la hora de considerar el impacto de los encaminadores analizados sobre las aplicaciones reales. Por otro lado, estas discrepancias en los resultados son obtenidas para dos de los encaminadores más sencillos desde el punto de vista de implementación, luego es previsible que al emplear estructuras más complejas el error se incrementará. Además de esta evidencia, existen otro tipo de detalles que tienden a reforzar esta aseveración, como es el hecho de que, si bien NETSIM está pensado para ser utilizado de forma modular y extensible, sus características, en lo que respecta al estilo de programación y facilidad de uso, son bastante discutibles. En pocas palabras, el desarrollo de cualquier propuesta en este entorno puede llegar a ser muy costoso en tiempo y con resultados poco precisos.

A.3.3 Implementación ED-SICOSYS.

A la vista de la clara inviabilidad de la propuesta anterior, el siguiente paso ha sido conjugar las características del sistema superior que emula RSIM con nuestro simulador SICOSYS. El modo de llevarlo a cabo ha sido reemplazar el modulo original encargado de simular la red de RSIM por nuestro propio simulador. El principal problema de este planteamiento ha sido el control de la simulación: por un lado SICOSYS es un simulador conducido por tiempo y gran parte de RSIM es un simulador conducido por eventos. Este inconveniente ha sido superado gobernando, desde el simulador SICOSYS, el driver de la simulación (YACSIM). La estructura de funcionamiento del simulador propuesto, y que hemos denominado ED-SICOSYS (*Execution Driven* SICOSYS), se muestra en la Figura A-13.

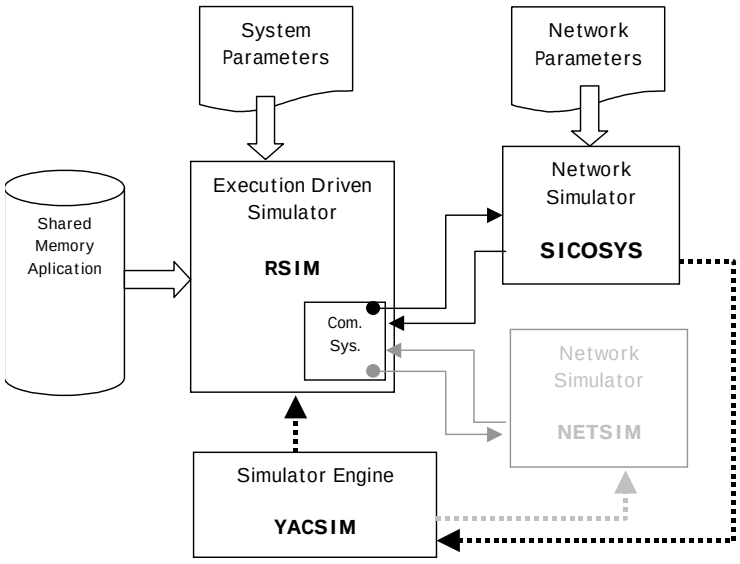

Figura A-13. Arquitectura de ED-SICOSYS.

Es conocido que la precisión que exhibe SICOSYS presenta como contrapartida un rendimiento notablemente inferior a simuladores como NETSIM. Este hecho hace que, globalmente, el sistema de simulación muestre un rendimiento en torno a 3 ó 4 veces peor que el simulador original, dependiendo estas cifras de la aplicación y del tamaño del problema considerado.

A continuación, seleccionaremos una aplicación prueba y la ejecutaremos sobre cada una de las alternativas de simulador conducido por ejecución: el que contiene NETSIM y el que contiene SICOSYS. En este análisis se ha empleado, como en el punto anterior, un encaminador determinista, con mecanismo de evitación de *deadlock* basado en *Burbuja*, sobre dos redes toro 4x4 y 8x8. En ambos casos se ha fijado la velocidad relativa, entre el reloj interno del procesador y el encaminador, a 2. En este análisis se han empleado redes aisladas de peticiones y datos con la configuración por defecto de RSIM en lo que respecta a la jerarquía de memoria y características del procesador (Ver Tabla 3-9 en la página 126). En la Tabla A-2 se muestran los resultados, tanto de tiempo de ejecución de la aplicación como del tiempo de simulación en un SGI Power Challenge con procesadores MIPS R10000 a 200Mhz, 2 Mb de cache de segundo nivel y 1Gbyte de RAM, sobre IRIX6.2.

Aplicación	RSIM		ED-SICOSYS	
	Tiempo de Ejecución. (ciclos)	Tiempo de Simulación. (seg.)	Tiempo de Ejecución. (ciclos)	Tiempo de Simulación. (seg.)
FFT 16 Procesadores	1131860	1665	932352	2716
FFT 64 Procesadores	650845	3767	519505	9413

Tabla A-2. Comparativa de rendimiento y precisión entre RSIM y ED-SICOSYS.

A la vista de estos resultados, queda claro que existe una discrepancia considerable en los tiempos de ejecución de la aplicación entre ambos simuladores, consecuencia directa de la mayor precisión alcanzada por SICOSYS. El error oscila, para esta aplicación, entre un 20% para el caso de 16 nodos y un 25% para el caso de 64 nodos. Sin embargo, el coste de la superior fiabilidad de SICOSYS se traduce en una notable caída de rendimiento en el tiempo de simulación. La caída de rendimiento es especialmente acusada en el caso de 64 nodos y es del orden de un 250%.

Frente a la caída del rendimiento del simulador propuesto y dado con que empleamos un sistema multiprocesador SMP como *host* a la hora de ejecutar las simulaciones, podemos afrontar el problema incorporando al simulador un cierto grado de paralelismo. En este caso, y dada la clara división que existe en la estructura interna del simulador, lo más sencillo es paralelizar la ejecución del simulador de tal forma que el simulador de la red de interconexión corra en un procesador y el resto del sistema se ejecute sobre otro diferente. En este caso, el problema se centra en la sincronización entre los dos procesos del simulador. Dentro de las diversas técnicas existentes a la hora de paralelizar un simulador de este tipo, nosotros hemos optado por la más conservadora: siempre que exista algún mensaje circulando por la red de interconexión será necesario sincronizar ambos procesos en cada uno de los flancos de reloj de la red de interconexión. En general, y dado que habitualmente la red tiene un reloj más lento que el procesador entre 2 y 4 veces, será preciso realizar la sincronización aproximadamente entre un 50% y 25% de los ciclos de ejecución de la aplicación. Pese a esto, el grano de paralelismo sigue siendo extremadamente fino. La sincronización se realiza a través de dos colas compartidas por ambos procesos que son las encargadas de almacenar los mensajes enviados hacia la red y recibidos desde la red. Cada proceso tiene permiso de escritura en una sola de las colas, como se muestra en la Figura A-14. En cada sincronización, los mensajes listos para enviar o que han llegado a destino serán introducidos en los inyectores e interfaces de red respectivamente.

No cabe duda que la comunicación entre procesos es de un grano extraordinariamente fino, por lo que es necesario ser muy cuidadoso con el método empleado para realizar la sincronización. En este caso las sincronización se realiza mediante una barrera, pero sobre el sistema evaluado no todas las metodologías muestran el mismo rendimiento. De hecho se ha probado el sistema con *pthreads* [92] y los *sprocs* nativos de IRIX, obteniendo resultados pésimos de rendimiento, dado que el coste de la sincronización supera, con mucho, las posibles ganancias de ejecución. La opción empleada finalmente ha sido implementar la barrera de sincronización entre los dos procesos a partir de un semáforo *adhoc* como se describe en [116] y el rendimiento alcanzado no consigue eliminar completamente la penalización introducida por SICOSYS, pero si mejorar el tiempo de simulación considerablemente. En concreto se logran ganacias de hasta un 30% en tiempo de simulación. En la Tabla A-3 se muestran los resultados alcanzados con esta estrategia de simulación y la original.

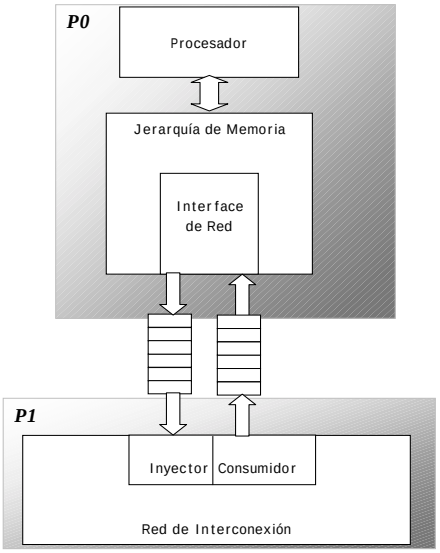

Figura A-14. Estructura del simulador EDD-SICOSYS (*Execution Driven Distributed SICOSYS*).

Aplicación	ED-SICOSYS		EDD-SICOSYS	
	Tiempo de Ejecución. (ciclos)	Tiempo de Simulación. (seg.)	Tiempo de Ejecución. (ciclos)	Tiempo de Simulación. (seg.)
FFT 16 Procesadores	932352	2716	932352	2639
FFT 64 Procesadores	519505	9413	519505	7126

Tabla A-3. Comparativa de rendimiento y precisión entre ED-SICOSYS y EDD-SICOSYS.

Las mejoras de rendimiento son notables cuando la red ha de manejar grandes cantidades de información o cuando es de un tamaño considerable. En este caso, se observa como para 16 nodos las diferencias en el tiempo de simulacion apenas si llegan a un 5%, fundamentalmente

debido a que el *overhead* de sincronización entre los dos procesos es casi tan costoso, en el porcentaje del tiempo de ejecución, como lo es en sí la simulación de la red. Sin embargo, en el caso de 64 nodos, las diferencias son bastante más apreciables llegando la mejora en tiempo de simulación de EDD-SICOSYS frente a ED-SYCOSIS de hasta un 25%, quedando las diferencias de rendimiento definitivas entre EDD-SICOSYS y RSIM en este caso en torno a un 190%. Por lo tanto, es preciso ser conscientes de la utilidad de esta implementación. Su utilización es más adecuada cuando el objetivo es comprobar el rendimiento de una posible propuesta en un encaminador determinado pero, si la idea es probar dos o más alternativas siempre va a ser más adecuado ejecutarlas en modo secuencial dado que el *throughput* de tareas siempre va a ser superior en estos casos.

A.4 Cargas Reales. Simulador Conducido por Trazas.

En general los sistemas de memoria compartida y más en particular las arquitecturas tipo ccNUMA, someten a la red a un tráfico que no está asociado a llamadas o primitivas como en los sistemas de paso de mensajes, sino que depende implícitamente de los accesos a memoria y del protocolo de coherencia empleado en el sistema, por lo que resulta muy compleja la obtención de resultados fiables del rendimiento. En principio, parece que las evaluaciones realistas únicamente pueden obtenerse a través de la ejecución simulada de la aplicaciones. Pero si esta aplicación corre sobre un sistema real, aparte de la imposibilidad obvia de modificar la red subyacente, la toma de datos afectará significativamente a los propios datos obtenidos. Consecuentemente, la única opción viable parece ser la simulación, prácticamente completa, del sistema multiprocesador aunque ello implique una cantidad de tiempo y recursos computacionales, en ocasiones, impracticables.

Tratando de aliviar este problema, en este punto del apéndice se introduce una metodología de simulación dirigida hacia la evaluación de las redes de interconexión de sistemas multiprocesador que relaja los requerimientos de la simulación conducida por ejecución. Con este planteamiento se pretende evitar la imprecisión asociada a las metodologías basadas en los tráficos de tipo sintético sin tener que recurrir a costosos simuladores conducidos por ejecución. Esta parte del apéndice se centra en efectuar un análisis comparativo entre dos metodologías de simulación y observar cuál es el impacto que presenta cada una de ellas sobre los resultados, tanto en precisión como en coste computacional. Las metodologías comparadas son: la simulación conducida por ejecución y una variación de la simulación conducida por trazas.

Los resultados alcanzados por la metodología propuesta para un sistema ccNUMA con procesadores ILP, justifica claramente su uso en el ciclo de diseño de los elementos de la red de interconexión. Estos resultados muestran errores promedio en torno 4% (el error máximo es inferior

al 10%) en el tiempo de ejecución de las aplicaciones y ganancias de velocidad superiores al 400% con respecto a la simulación conducida por ejecución. A la hora de valorar el significado del último dato, es necesario tener en cuenta los efectos multiplicativos que tiene: el proceso de diseño requiere de la realización de varias simulaciones cuando se está explorando el espacio de diseño, es por ello que la ganancia en el tiempo de ejecución se multiplicará tantas veces como sea preciso volver a simular el sistema. La metodología no se presenta como una alternativa a la simulación conducida por ejecución, sino como un paso intermedio a la hora de explorar el espacio de diseño. Desde este punto de vista, la simulación conducida por ejecución es oportuna exclusivamente en los últimos estadios del ciclo de diseño.

A.4.1 La Simulación Conducida por Traza en el Análisis de la Red de Interconexión.

En el diseño de las redes de interconexión de sistemas multiprocesador es necesario el empleo de *workloads* que representen lo más fielmente posible las exigencias a las que será sometida la red cuando en el sistema se ejecuten las aplicaciones paralelas. En las primeras fases del diseño, el tipo de cargas de trabajo empleadas son, fundamentalmente, cargas de tipo sintético. Esta clase de tráfico presenta unas características bien conocidas que permiten realizar fácilmente comparativas con otras redes y/o otros estudios. No obstante, el empleo de este tipo de cargas implica serias limitaciones, ya que solo permiten modelar el comportamiento de algunas aplicaciones reales de forma muy aproximada.

Una segunda alternativa, que se aproxima bastante más a la realidad, consiste en la utilización de trazas de aplicaciones reales obtenidas a partir de máquinas reales. Instrumentalizando la ejecución de la aplicación paralela sobre un sistema real se obtienen los mensajes generados por cada procesador. Esto permite alimentar un simulador del subsistema de comunicación y determinar el rendimiento de éste ante el tráfico real asociado a la aplicación instrumentalizada. El tiempo necesario para la obtención de las trazas puede ser apreciable, pero posteriormente podrán ser reutilizadas durante gran parte de la fase de diseño. Este tipo de estrategias está muy extendido en la evaluación de rendimiento de diversas áreas de los computadores. Es habitual su uso en el estudio de la jerarquía de memoria de sistemas uniprocesador.

El empleo de este tipo de estrategias esta limitado por las características intrínsecas del sistema evaluado. Es claro, por ejemplo, que su empleo en la evaluación de rendimiento de la jerarquía de memoria es correcto para el caso de procesadores simples, sin embargo el utilizar técnicas de este tipo para determinar el comportamiento de la jerarquía de memoria en procesadores ILP con ejecución fuera de orden, accesos a memoria no bloqueantes, etc, es obviamente poco realista y puede dar lugar a errores considerables [80]. Sin embargo, en arquitecturas de paso de

mensajes, el empleo de una estrategia basada en simulación conducida por trazas puede permitir evaluar la red de interconexión con una precisión aceptable. Con este paradigma de programación, es posible obtener una relación de orden causa-efecto para cada uno de los eventos generados por la aplicación. Además el orden en que se producen es siempre determinista, dado que las dependencias aparecen siempre de forma explícita en el nivel de aplicación. Existe una gran variedad de trabajos en los que el estudio de la red de interconexión se aborda desde este punto de vista [32].

De diferente modo, en el caso de los procesadores ILP, en un entorno de arquitectura ccNUMA con aplicaciones de memoria compartida, analizar el impacto de la red sobre las aplicaciones mediante esta aproximación no parece adecuado en absoluto. La causa fundamental de este hecho es que el estado o características de la propia red influyen directamente sobre las futuras referencias. Así por ejemplo, una red más o menos cargada influirá sobre el número de mensajes generado por el protocolo de coherencia. Incluso la propia instrumentalización para extraer la traza influirá sobre los datos que se obtengan. Si a esto se le añade los efectos de emplear ILP en los nodos de proceso, se entiende que esta metodología no sea fiable para este tipo de arquitecturas.

La solución pasa por llevar a cabo simulaciones conducidas por ejecución, como hemos introducido en el punto anterior. En este caso, el simulador del subsistema de comunicación recibe una referencia o mensaje desde el simulador del sistema multiprocesador, simula el camino a lo largo del subsistema (incluyendo la contención) y retorna el tiempo empleado para alcanzar el destino. Esto permite al simulador del sistema multiprocesador avanzar el reloj para generar las siguientes referencias emulando completamente el comportamiento de la máquina estudiada, es decir, simulando completamente los procesadores y el resto de la jerarquía de memoria.

La precisión de los resultados de la simulación conducida por ejecución es muy elevada. Sin embargo, los costos computacionales requeridos, y sobre todo, el tiempo necesario para llevar a cabo todas las simulaciones que requiere la fase de diseño completa de un encaminador o red pueden llegar a hacer su uso impracticable, incluso con sistemas de unas pocas decenas de nodos. Existe un gran número de trabajos que intentan reducir el tiempo de simulación de este tipo de estrategias como es la ejecución directa [47][109]. En estos trabajos se aportan soluciones a los costes de la simulación de un sistema ccNUMA con procesadores ILP, suprimiendo parcialmente la simulación de los procesadores. Es claro que otra posible solución es emplear procesadores simples en la simulación emulando el comportamiento de ILP sin mas que multiplicar su frecuencia de reloj por el número de vías del procesador a simular[66]. Sin embargo, como se muestra en [47] los errores a que puede dar lugar este planteamiento son muy altos.

A.4.2 Simulación Conducida por Trazas con Realimentación desde la Red (TDS-NF).

Esta fuera de toda duda que, en las primeras fases del diseño, las mejores pruebas para el análisis del rendimiento del subsistema de comunicación pasan por la utilización de tráficos sintéticos por su facilidad, sencillez y claridad. No obstante, también esta claro que posteriormente será necesario utilizar tráfico real para válidar cualquier posible modificación o propuesta.

La metodología propuesta en este trabajo intenta aunar las mejores características de cada una de las dos estrategias de simulación, es decir, se pretende alcanzar las características de velocidad de la simulación conducida por traza, con una precisión similar a la alcanzada con la simulación conducida por ejecución (Ver Figura A-15). Esta metodología la hemos denominado *Simulación conducida por Trazas con Realimentación desde la red (Trace-Driven Simulation with Network Feedback TDS-NF).*

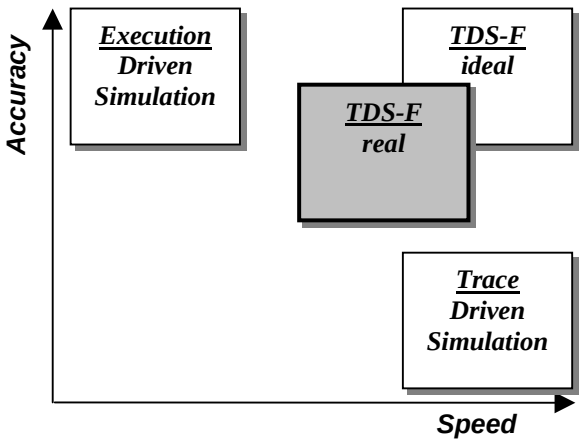

Figura A-15. Relaciones de velocidad y precisión con cada uno de los planteamientos de la simulación.

La idea básica es simple: obtener una traza de los mensajes generados por un sistema ejecutando aplicaciones reales y eliminar, en la medida de lo posible, toda dependencia de la red de interconexión empleada en el instante de realizar la extracción de la traza. Esto se lleva a cabo eliminando toda dependencia temporal absoluta en la generación de eventos y relativizando los instantes de envío de mensajes a las condiciones de la red en cada momento. Así por ejemplo, en la Figura A-16 se ha representado el envío, desde un proceso P_1 a otro P_0, de un mensaje m_2 como respuesta a un mensaje m_1 recibido previamente. En la *Figura A-16.a* el envío de m_2 comienza t unidades de tiempo después de comenzar la ejecución de la aplicación y en la Figura A-16.(b), m_2 es enviado t_C unidades de tiempo después de transcurrido el tiempo t_N de transferencia por la red. Recoger en una traza el tiempo t y emplearlo para lanzar la siguiente referencia, como en el primer caso, implica no poder aislar la influencia de la red empleada en la

instrumentalización, mientras que t_C permite relativizar el envío del mensaje m_2 a un instante que depende del estado actual de la red. Por ello, en cierta medida, las referencias futuras de la traza dependen del estado de la red, es decir, el flujo de información entre el generador de referencias y el simulador de la red de interconexión es bidireccional no como ocurre en las simulaciones conducidas por trazas convencionales, donde la información solo tiene un sentido: desde el generador de referencias hacia el simulador del subsistema bajo estudio. La realimentación entre los dos sistemas permite establecer, como figura de mérito de la evaluación, el tiempo de ejecución de la aplicación sin ningún problema.

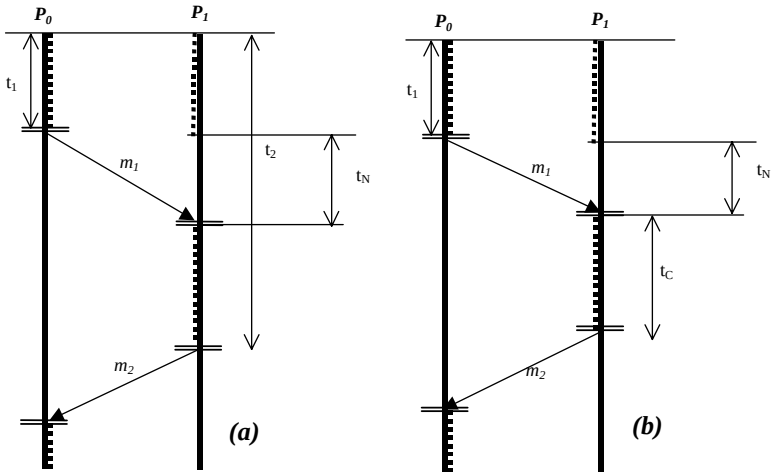

Figura A-16. Temporalización absoluta (a) y relativa (b).

Como se puede apreciar, la idea básica se reduce a eliminar de la traza toda posible influencia de la red de interconexión y marcar exclusivamente las características de los tiempos de ocupación de los procesadores. Los tiempos de espera entre recepción y envío de mensajes, corresponden al tiempo que emplea el nodo de proceso en operaciones independientes del subsistema de comunicación.

Como es obvio, esta metodología sólo es válida para establecer el rendimiento de la red de interconexión, en otras palabras cualquier modificación tanto en la arquitectura del procesador como en la jerarquía de memoria, inutiliza las trazas a la hora de medir el rendimiento de la red de interconexíon.

A.4.2.1 Particularización para el caso ccNUMA.

Dependencias entre mensajes.

En una arquitectura ccNUMA, la extracción de la información correlacionando los distintos mensajes a través de los parámetros de disparo, tal y como ha sido presentado anteriormente, no es sencilla dado que no es la aplicación la que genera los mensajes directamente, sino el sistema de memoria. Además, la capacidad de los procesadores ILP con accesos a memoria no bloqueantes complica aún más el problema, dado que la correlación entre mensajes no es inmediata.

Para atajar este problema se recurre a simplificar el sistema de tal manera que el orden causal de los envíos y recepciones de la traza original se conserve. Para ello basta con relativizar cada uno de los envíos, con la recepción inmediatamente precedente en el nodo fuente. Esta simplificación puede ejemplificarse en la Figura A-17. En este caso el mensaje m_{th} dispara el envío de tres mensajes desde el nodo P_3 hacia el nodo P_2 y P_1. Si suponemos que m_{th} se trata de un mensaje de petición de una línea de almacenada en P_3 y m_1 representa el dato solicitado por P_2 es claro que el modelado es preciso, ya que el tiempo de espera de este mensaje modelará el tiempo de acceso al directorio de P_3 y la generación del dato solicitado por P_2. Sin embargo, es claro que los restantes mensajes pueden no tener relación alguna con m_{th} al poder tratarse de mensajes de petición de otros datos o respuestas a peticiones previas.

El carácter reactivo de parte del tráfico que circula por la red, en este tipo de arquitecturas, permite que el planteamiento anterior pueda refinarse aún más relativizando el envío de los mensajes, exclusivamente, con respecto a la recepción de mensajes de petición. De esta forma, cada mensaje respuesta de origen *s* y destino *d* estará supeditado a la recepción de la ultima petición asociada que llegó a *s* desde el nodo *d.*

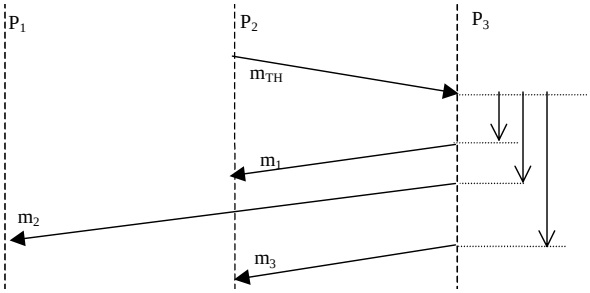

Figura A-17. Aproximación inicial.

Sin embargo, el problema no resuelto es el de los mensajes de petición: en este tipo de mensajes es imposible, sin simular el sistema superior del nodo, determinar cuál es la causa que los pro-

vocó y más aún correlacionarlo con alguna recepción por parte del nodo generador. La aproximación burda empleada con este tipo de mensajes ha sido la de suponer que estarán disparados por el último mensaje de respuesta recibido por el nodo local.

Tipo de Mensaje Recibido.	***Tipos de Mensaje Disparados.***	
READ_SH	REPLY_SH	DATA
	REPLY_EXCL	COHE
	COPYBACK	
READ_OWN	REPLY_EXCL	DATA
	RELPY_EXCLDY	
	COPYBACK_INVL	COHE
	INVL	
UPGRADE	RELPY_EXCLDY	DATA
	REPLY_EXCL	
	REPLY_UPGRADE	COHE
	INVL	
any REPLY	READ_SH / READ_OWN / UPGRADE	

Tabla A-4. Relaciones de disparo entre mensajes.

Teniendo en cuenta estos criterios puede establecerse como es la correlación, en primera aproximación, entre mensajes entrantes y salientes. Obviamente, para fijar dicha correlación es imprescindible conocer el protocolo de coherencia de la máquina en cuestión. Así, para el caso bajo análisis y considerando un protocolo de coherencia basado en inválidación MESI, se establece el conjunto de relaciones entre mensajes mostrado en la Tabla A-4. Como es lógico, además de estas relaciones en los mensajes de respuesta y el par origen-destino, es necesaria la correlación con las direcciones de memoria a la que se refieren.

Estructura de las entradas de la traza

La traza extraída de la ejecución de las aplicaciones paralelas deberán, por una parte, poseer la independencia del estado actual de la red de interconexión empleada para su extracción y por otra parte, portar las relaciones señaladas en la tabla anterior. Para ello, cada una de las entradas de la traza contiene los campos habituales (origen, destino, tamaño) y cuatro campos adicionales que, posteriormente, durante el empleo de la traza, viajarán con el mensaje por la red hasta

su destino. Estos cuatro campos son: la *etiqueta* del mensaje, la *etiqueta de disparo* del mensaje, el *tiempo de espera* posterior al disparo y el *número de mensajes disparados*.

Origen	Destino	Etiqueta	Tamaño	Etiqueta del Disparo	Tiempo de Disparo	Número de Mensajes Disparados

Tabla A-5. Estructura de las entradas del fichero de traza.

El campo *etiqueta del disparo* indica que el mensaje no podrá estar listo para enviar hasta que no se reciba algún mensaje con esa *etiqueta*. La etiqueta corresponde con la dirección de memoria a la que se refiere el mensaje. Una vez se haya recibido ese mensaje, aún deberá transcurrir el número de ciclos indicado por el campo *tiempo de disparo* antes de inyectarlo en la red. Finalmente, el *número de mensajes disparados* indicará el número de mensajes que pasaran a estar listos cuando el mensaje sea recibido en destino.

A.4.3 Válidación de la metodología propuesta.

A.4.3.1 Extracción de la Traza.

En el punto previo se han expuesto los problemas asociados al emplear esta metodología en el caso de los sistemas ccNUMA con procesadores ILP. A priori, parece que las aproximaciones realizadas, especialmente con los mensajes de petición, son bastante toscas y que los resultados no van a justificar su aplicación. Sin embargo esto no es así y en este apartado nos centraremos en demostrarlo. Para llevar a cabo el análisis de la metodología partiremos de un sistema bien conocido, empleando aplicaciones realistas.

El sistema empleado para la extracción de la traza, mostrado en la Figura A-18, está basado en el simulador conducido por ejecución presentado en la seccion Sección A.3

A partir de este sistema de simulación, es posible extraer la traza de las aplicaciones sin más que capturar las referencias lanzadas por cada interface de red. Durante esta fase del análisis, cada traza constituida por eventos de envío y recepción, es post-procesada con el fin de eliminar los efectos de la red de interconexión empleada en su extracción y convertirla en una estructura de datos cuyas entradas tengan un formato como el señalado en el apartado anterior.

.

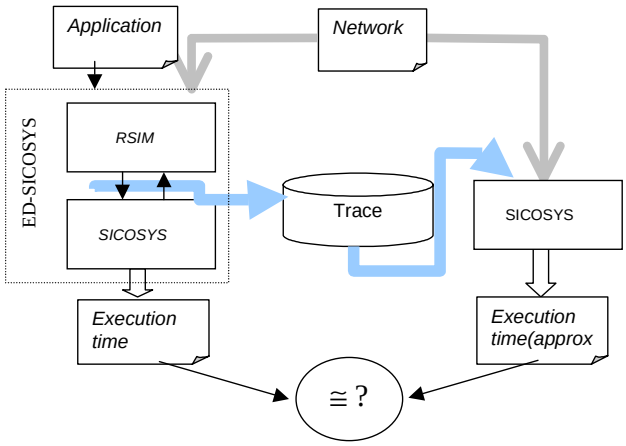

Figura A-18. Fase 1: extracción de la traza y validación de resultados.

Una vez extraída la traza y tras el post-procesado necesario, la validación de los resultados se lleva a cabo en dos fases. En la primera fase se alimenta el simulador SICOSYS con la traza post-procesada utilizando la misma red de interconexión que en la extracción. Los resultados de tiempo de simulación y de tiempo de ejecución de la aplicación son comparados con la simulación conducida por ejecución.

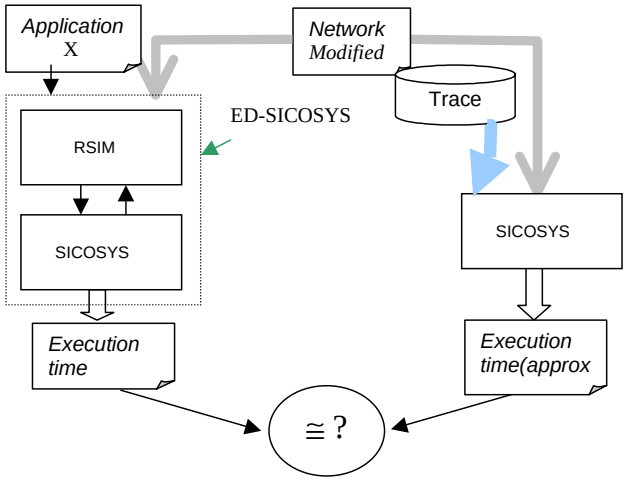

Figura A-19. Aplicación de la metodología.

La fase siguiente consiste en analizar la ejecución de la misma aplicación alterando, en mayor o menor medida, las características de la red de interconexión (Figura A-19). Para ello se emplea la traza obtenida previamente y se repiten las simulaciones conducidas por ejecución para cada red considerada. De la misma forma que en la fase anterior se comparará cada resultado, tanto en tiempo de simulación con cada estrategia, como en tiempo de ejecución de la aplicación.

Como es obvio, todos los análisis comparativos se centrarán en las fase de ejecución centrales de la aplicación, es decir, se obviarán las fases de inicialización y finalización.

A la hora de comparar las velocidades relativas de cada metodología, todas las simulaciones han sido realizadas en un SGI Power Challenge LX, con procesadores R10000 a 200 MHz, 1 GB de RAM y 2 MB de cache L2. Los dos simuladores han sido compilados con el compilador de C/C++ MIPSPro 7.2 y empleando opciones de máxima optimización en la compilación.

A.4.3.2 Resultados de la metodología: FFT y Radix.

La metodología expuesta previamente ha sido utilizada sobre las aplicaciones FFT y Radix, pertenecientes a la suite SPLASH2[137]. La elección de estas dos aplicaciones viene motivada por que son de las que más sensibilidad muestran a las características de la red de interconexión como hemos mostrado en la Sección 2.3.3 en la página 53. Los experimentos se han separado en dos configuraciones diferentes: por un lado, redes de 16 nodos y por otro, de 64. El tamaño del problema empleado, en el caso de FFT, consiste en 64K complejos dobles. En el caso de Radix, el problema esta constituido por un conjunto de 512K claves enteras a ordenar, con un radio máximo de 512K.

La configuración del sistema procesador empleado ha sido, básicamente, la establecida por defecto en el simulador RSIM, aunque se emplean procesadores a 600 MHz y el tamaño de las líneas de cache, tanto L1 como L2, es de 32 bytes. El tamaño de los comandos básicos es de 8 bytes.

La configuración de la red empleada en la extracción de la traza ha sido una malla con encaminamiento adaptativo y unificada para el tráfico de peticiones y respuestas. Los canales de comunicación tienen una anchura de 4 bytes y la velocidad de la red es de 275Mhz. El tamaño de los bufferes de tránsito es de 4 paquetes de 10 phits por canal virtual y el control de flujo empleado es *Virtual Cut-Throught* (VCT). Se emplean 2 canales virtuales por línea física: un canal virtual completamente adaptativo y otro determinista. La evitación de deadlock por desbordamiento de los canales de consumo se elimina incorporando a los puertos de consumo, en las interfaces de red, de suficiente capacidad de almacenamiento para cada número de nodos y limitando las referencias lanzadas a la red en caso de detección de posible desbordamiento.

Bajo estas condiciones, el número de mensajes que constituyen la traza son los que se muestran en la Tabla A-6.

Radix		FFT	
16 Nodos	64 Nodos	16 Nodos	64 Nodos
564189	1220718	311985	342107

Tabla A-6. Número de mensajes generados por aplicación y número de procesadores.

Utilizando la traza generada bajo las condiciones señaladas, se han ido modificando diferentes características de la red de interconexión y comprobando los tiempos de ejecución de las aplicaciones y validándolos, mediante el simulador conducido por ejecución, de la forma señalada en el apartado anterior.

Las configuraciones alternativas para la red de interconexión se han restringido a tres:

- Empleo de un toro adaptativo en lugar de la malla. El mecanismo de evitación de deadlock está basado en la burbuja.
- Reducir el ancho de los canales de comunicación, entre routers, a 2 bytes.
- Duplicar la velocidad de reloj de los encaminadores de la red original.

Todas las modificaciones pueden calificarse de severas, dado que todas ellas afectan notablemente al tiempo de ejecución de las aplicaciones. En cierta medida, al elegir estas tres opciones sobre aplicaciones que hacen un uso intensivo de la red de interconexión, estamos determinado cuál es el error de peor caso cometido por la metodología.

Los resultados obtenidos para cada configuración y red, con cada simulador, se han comparado tanto en las diferencias de tiempo de simulación como en el error cometido por la simulación conducida por trazas con respecto a la simulación conducida por ejecución. Los datos, para cada una de las aplicaciones y configuraciones, se muestran en las figuras siguientes.

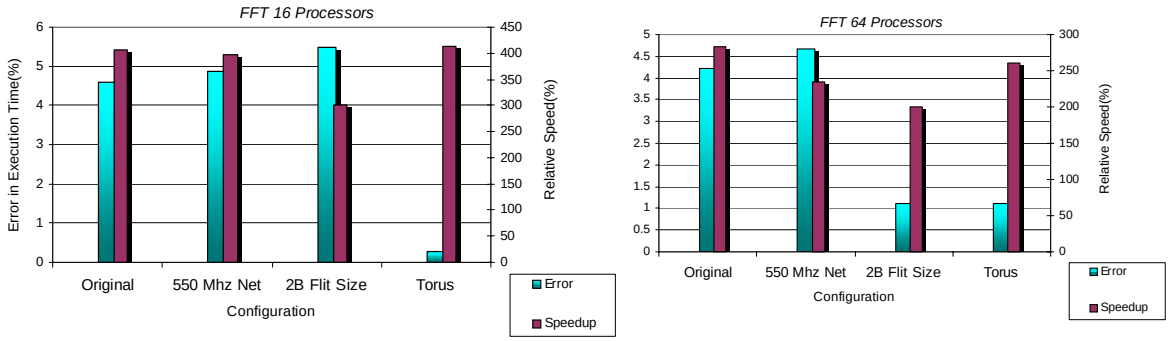

Figura A-20. Error relativo en valor absoluto entre la simulación conducida por trazas y por ejecución y ganancia en velocidad para FFT.

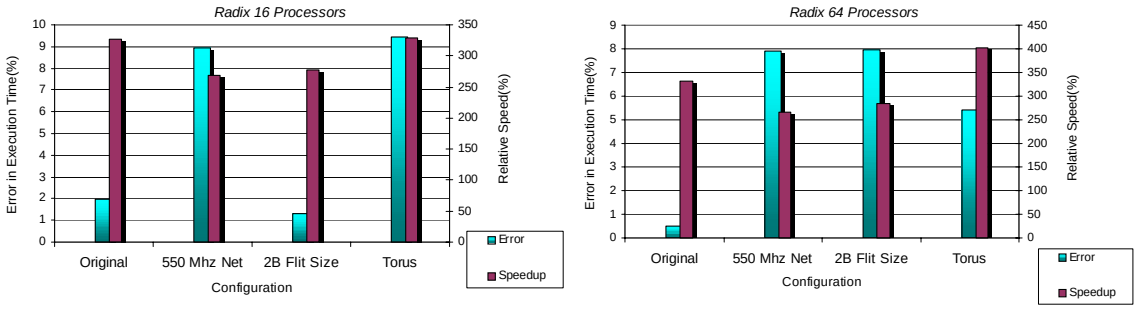

Figura A-21. Error relativo en valor absoluto entre la simulación conducida por trazas y por ejecución y ganancia en velocidad para Radix.

Configuration	FFT16		FFT64		RADIX16		Radix 64		***Average***	
	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)
Original	4.6	407	4.23	283	-1.99	328	0.37	332	**2.8**	**338**
Net. Speedx2	-4.85	397	4.67	235	8.94	268	7.93	265	**6.59**	**291**
Flit Size/2	5.48	302	1.1	200	1.33	278	7.97	283	**3.97**	**265**
Torus Network	0.28	412	-1.1	261	9.45	330	5.76	402	**4.14**	**351**
Average	**3.80**	**379**	**2.75**	**245**	**5.45**	**301**	**5.55**	**396**	***4.37***	***311***

Tabla A-7. *Resultados de Error y Speedup (promedios en valor absoluto).*

Los resultados muestran diferencias, en los tiempos de ejecución entre la simulación conducida por trazas y la simulación conducida por ejecución, siempre inferiores al 10%. Como se puede

apreciar en la Tabla A-7, el error promedio cometido apenas supera el 4%. Estos resultados indican un grado de precisión que, a priori, no parecía alcanzable dadas las aproximaciones propuestas de partida. El error se mantiene acotado para las configuraciones comparadas pese a que la diferencia de tiempo de ejecución entre cada una de las configuraciones es considerable. Por ejemplo, la diferencia entre las redes con 2 bytes de ancho de *flit* y la red con un reloj el doble de rápido que la configuración original se observan diferencias superiores al 50% en tiempo de ejecución de algunas aplicaciones. En cuanto a la configuración que implica un error máximo se observa como los casos de redes con mayor rendimiento que la considerada en la extracción de la traza, el error es mayor. Este hecho es más acusado en el caso de FFT. Fundamentalmente, el origen del error está en la correlación conservadora de los mensajes de petición, dado que en muchos casos no se modela adecuadamente los accesos remotos no bloqueantes. Esto trae como consecuencia que, en la mayoría de los casos, el tiempo de ejecución de la aplicación empeore pese a que la red tiene mejores prestaciones. Los efectos de este hecho son mayores cuanto mayor sea el número de mensajes generados por la aplicación. Por otro lado, el error promedio tiende a conservarse al variar el número de nodos de la aplicación.

La ganancia en tiempo de simulación de la simulación conducida por trazas frente a la conducida por ejecución para cada caso, oscila entre un 200% y 400%. Aunque el coste computacional del procesado dinámico de la traza, para la búsqueda de la dependencia entre mensajes, produce una cierta penalización del tiempo de simulación, es muy significativa la reducción temporal de cada experimento. A modo de ejemplo, cabe citar que el tiempo de simulación para el caso de Radix y una red de 64 nodos, sobre el sistema empleado para las simulaciones, pasa de ser aproximadamente de 11 horas para la simulación conducida por ejecución, a tan solo 3h. en la simulación conducida por trazas. Si, además, se tiene en cuenta el elevado número de experimentos que es necesario llevar a cabo en un proceso normal de diseño, estas ganancias tendrán un efecto multiplicativo.

Por último, es importante señalar que la precisión alcanzada por SICOSYS requiere un tiempo de simulación considerablemente más alto que simuladores funcionales más sencillos. Es por ello que, pese a no simular ni la jerarquía de memoria de cada nodo ni sus procesadores, la disminución en el tiempo de ejecución no sea mayor. Obviamente, el empleo de simuladores para la red de interconexión mas simples implicaría diferencias en el tiempo de simulación mucho más altas, pero también mayores errores.