SimuRed: Un simulador de redes de multicomputadores interactivo y visual

Fernando Pardo y Jose A. Boluda

Resumen— **En este artículo se presenta SimuRed, un simulador capaz de evaluar diversas configuraciones de redes de multicomputadores de forma gráfica e interactiva. El principal objetivo de la herramienta es educacional por lo que intenta mostrar el tráfico de paquetes a lo largo de la red, bloqueos, etc. de una manera sencilla y gráfica. Adicionalmente el núcleo del simulador es suficientemente flexible para que pueda ser ampliado de manera que se pueda utilizar en experimentos de investigación con la intención de evaluar las prestaciones de un abanico amplio de redes de multicomputadores. Todos los programas y el código fuente están libremente disponibles en la página web del simulador.**

Palabras clave— **Redes de multicomputadores, simulación.**

I. Introducción

La mayoría de simuladores de redes de multicomputadores han sido diseñados, y tradicionalmente usados, para investigación [1]. Por esta razón, la mayoría de ellos no son de uso muy amigable y/o carecen de una interfaz gráfica [2], [3], [4]. SimuRed ha sido diseñado con el objeto de ser una herramienta con alto valor pedagógico sin perder por ello precisión ni fiabilidad en sus cálculos. SimuRed es sencillo de utilizar, disponiendo adicionalmente de interfaz gráfica y capacidades de visualización del estado de la red y sus elementos.

Una de las características más interesantes de SimuRed es que ofrece la posibilidad de realizar simulaciones interactivas, es decir, permite visualizar la estructura de la red y el tráfico de paquetes a nivel de flit, además de tener la posibilidad de avanzar paso a paso en la simulación, pudiéndola detener en cualquier instante [5]. La interfaz del simulador ofrece adicionalmente la posibilidad de simular en modo segundo plano, de manera que es posible especificar una simulación múltiple cambiando hasta dos parámetros para obtener un conjunto de gráficas que pueden visualizarse utilizando la propia interfaz gráfica del simulador (obsérvese la figura 4). También es posible utilizar otras herramientas de generación de gráficos pues los resultados se almacenan en el formato estándar de texto CSV, que puede ser leído por muchas herramientas gráficas. La inclusión de una herramienta de visualización de gráficos en el propio simulador permite ver los resultados de forma inmediata acelerando la posibilidad de realizar nuevas simulaciones.

El simulador está compuesto por una interfaz visual (*Front-End*) y un núcleo que es el que realiza las simulaciones. Ambas partes están perfectamente diferenciadas y son completamente independientes entre sí, aunque a nivel de usuario esta diferencia es imperceptible pues se integran perfectamente una dentro de otra.

El núcleo de SimuRed está formado por un conjunto de clases y métodos en C++ que pueden integrarse dentro de cualquier aplicación hecha a medida para la evaluación de prestaciones de una red de multicomputadores. El código fuente puede compilarse en cualquier plataforma con a priori cualquier compilador de C++. En la actualidad hay disponibles binarios completamente funcionales para Windows y Linux. En cuanto al interfaz de usuario se incluyen dos: una para Windows que funciona también en Linux completamente visual y gráfica, que es la que se discute en este artículo, y otra menos amistosa basada en línea de comandos que funciona tanto en DOS como en cualquier interprete de comandos de Linux. Las fuentes del núcleo y del interfaz en línea de comandos están libremente disponibles.

II. Modelado de la red

Es importante conocer el modelado que se ha hecho de la red de interconexión en el simulador para ver qué se puede simular realmente y cuáles son las limitaciones. La red se ha modelado en C++ intentando que este modelo sea lo más modular y ampliable posible, haciendo que las clases y métodos que definen la red y su comportamiento se parezcan lo más posible a sus equivalentes físicos y lógicos reales.

Una red de interconexión se puede definir a partir de tres elementos o capas [6]: topología, conmutación y encaminamiento. Las capacidades y limitaciones del simulador en cada una de estas capas se describen a continuación.

A. Topología de red

La mayoría de de las redes utilizadas para multicomputadores son redes con una topología estrictamente ortogonal. La topología de red que implementa SimuRed es el n-cubo k-ario (toro). Esta topología es estrictamente ortogonal y resulta ser la topología a partir de la cual se pueden construir otras que no son más que casos particulares como los hipercubos e incluso las mallas, que estrictamente hablando no son redes n-cubo k-arias pero que en determinadas circunstancias se pueden considerar como tales tal y como se muestra más adelante. Un hipercubo es un caso particular de n-cubo k-ario donde $k = 2$, de esta manera cualquier hipercubo n-dimensional es en realidad un n-cubo binario. El simulador muestra en cada momento la red para las dimensiones 0 y 1 tal

Dpt. Informática, ETSE, Universidad de Valencia, Avda. Vicente Andrés Estellés s/n, 46100 Valencia, España. {Fernando.Pardo, Jose.A.Boluda}@uv.es http://tapec.uv.es/

como se aprecia en la figura 1.

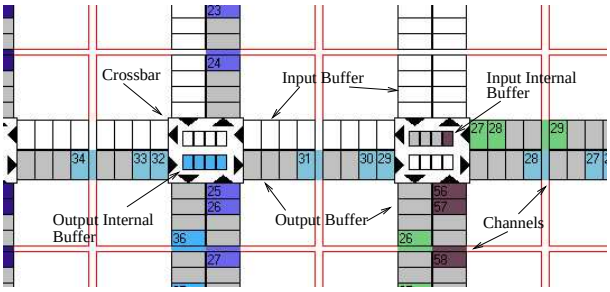

Fig. 1. Sección de la ventana del simulador donde se muestra la red junto con la situación de los paquetes.

Una malla es como una red n-cubo k-aria a la que se le ha eliminado el *canal de vuelta* en cada una de las dimensiones de la red. En SimuRed las mallas se implementan precisamente en el algoritmo de encaminamiento en lugar de en la topología misma, es decir, la topología tiene estos canales de vuelta, pero el algoritmo de encaminamiento no los usa, por lo que la simulación ocurre como si tuviéramos una malla en lugar de un toro. De esta manera SimuRed acepta cualquier topología de red ortogonal siempre que el número de nodos por dimensión (k nodos) sea constante. Para otras topologías más complejas se puede utilizar el mismo truco que para mallas, es decir, se puede simular cualquier topología siempre que *quepa* dentro del n-cubo k-ario, es decir, que sus canales de interconexión sean un subconjunto del toro.

Los canales de la red pueden ser unidireccionales o bidireccionales, siempre que lo soporten los algoritmos de encaminamiento. Se pueden definir varios canales virtuales para cada canal físico. Incluso es posible decidir que los canales virtuales sean todos ellos físicos en lugar de que compartan un mismo canal físico.

Estos son los parámetros que se pueden modificar en el simulador (se ha incluido en mayúsculas el parámetro modificable de la interfaz de usuario de línea de comandos):

Dimensiones. Son las dimensiones de la red. En principio no hay límite. Debe ser siempre mayor que cero. (DIMENSIONS).

Nodos por dimensión. Son los nodos en cada dimensión. El número total de nodos de la red es este número multiplicado por el número de dimensiones. Este número es fijo por lo que no se pueden implementar redes que tengan diferente número de nodos en dimensiones diferentes, es decir, sólo se pueden implementar redes n-cubo k-arias. Debe introducirse un valor mayor que cero. (NODOSDIM).

Longitud del buffer. Esta es la longitud de las colas Fifo de entrada y salida por cada canal. Tanto las colas de entrada como las de salida tienen la misma longitud, por lo que no pueden asignarse longitudes diferentes a unas y otras. El valor debe ser mayor que cero. (LONBUFFER).

Canales virtuales. Cada canal puede subdividirse en varios canales. Por defecto estos canales serán virtuales, es decir, comparten el mismo cable físico pero cada uno tiene su propio buffer, por lo que funcionan como si fueran físicos pero con el aumento de la latencia que conlleva compartir el mismo cable. Si realmente se quiere que estos canales sean físicos, hay una opción para ello (ver un poco después en este mismo punto). (NUMVIRT).

Bidireccional. Se puede especificar si los canales son bidireccionales o no. Hay que tener cuidado pues es posible que algunos algoritmos de encaminamiento no puedan encontrar un camino si el canal es unidireccional (el de orden de dimensión para mallas por ejemplo). (NUMDIR).

Adelanto en buffer. En una cola Fifo sencilla los flits van pasando de una posición a la siguiente, pero si hay posiciones vacías en la cola se puede adelantar el flit hasta justo antes de la siguiente posición ocupada, o directamente a la salida de la cola si ésta está vacía. Este adelanto es la opción por defecto, ya que es como funcionan normalmente las redes para obtener un mayor rendimiento. (FORWARDING).

Canales físicos. Si se selecciona esta opción entonces los canales virtuales son realmente físicos, es decir, cada canal virtual tiene su propio cable y por lo tanto la transmisión se realiza en paralelo junto con el resto de canales virtuales. (PHYSICAL).

B. Conmutación

SimuRed utiliza un modelo de encaminador general y sencillo: los elementos básicos se pueden apreciar en la figura 1 y son las colas Fifo de entrada/salida y el conmutador de barra cruzada (*Crossbar*). Todas las colas, tanto de entrada como salida, tienen el mismo tamaño expresado en flits que es la unidad básica de conmutación para el simulador. Los conmutadores de barra cruzada conectan cualquier entrada con cualquier salida libre. La conmutación de lombriz (*Wormhole Switching*) es el único mecanismo de conmutación implementado hasta el momento. Otras técnicas de conmutación encauzadas como la conmutación de paquetes o el *Virtual Cut-Trough* pueden emularse utilizando colas del tamaño adecuado y con ligeras modificaciones del código. Otras técnicas de conmutación no encauzadas como la conmutación de circuitos o técnicas más sofisticadas como la conmutación de exploración requieren un cambio más profundo en el modelo y el código.

Se han implementado cuatro **retrasos** en la red. Estos retrasos vienen expresados en ciclos de reloj:

Buffer. Es el tiempo necesario para pasar de una posición de las colas Fifo a la siguiente. Normalmente este tiempo es menor que el resto, por lo que no se suele incluir en muchos simuladores; no porque sea despreciable, sino porque el tiempo de transmisión del paquete, al realizarse de forma segmentada, depende del máximo de este tiempo y del tiempo de atravesar el *Crossbar*, y siempre suele ser este último mayor que el de la

cola Fifo. (DELFIFO).

Crossbar. Una vez establecido el camino interno en el encaminador, este tiempo es el que tarda un flit en atravesar el conmutador (*Crossbar*). (DELCROSS).

Conmutación. Es el tiempo que se tarda en establecer el camino dentro del encaminador, es decir, el tiempo que se tarda en decidir qué camino tomar y configurar el conmutador para ello. (DELSWITCH).

Canal. Es el tiempo que necesita un flit para atravesar el canal físico. (DELCHANNEL).

C. Encaminamiento

En SimuRed es sencillo implementar nuevos algoritmos de encaminamiento. De hecho, la herramienta dispone de varios algoritmos implementados. Los algoritmos de encaminamiento pueden restringir la utilización de canales para simular otras topologías distintas a los n-cubos k-arios. Un ejemplo es la simulación de mallas. De hecho, la mayoría de los algoritmos de encaminamiento implementados en la herramienta son para mallas en lugar de toros. Los algoritmos implementados son:

Orden de dimensión para mallas: Este es el algoritmo de encaminamiento libre de bloqueos para mallas más sencillo. Los paquetes se envían en una dimensión hasta que la distancia en esa dimensión es cero, entonces el paquete se envía a la dimensión inmediatamente superior donde se hace lo mismo y así sucesivamente. En el caso de que haya varios canales virtuales, se selecciona cualquier canal virtual libre siguiendo una selección aleatoria. Los canales deben ser bidireccionales pues de lo contrario no siempre sería posible establecer un camino y la red se bloquearía. (ROUTING=0).

Orden de dimensión para toros: Se trata del algoritmo anterior pero en éste se permite el uso de los canales de vuelta del toro. Este algoritmo no está libre de bloqueos mortales. Se ha incluido en el simulador para mostrar configuraciones de interbloqueo en la red. También es un algoritmo que funciona con canales unidireccionales aunque esto no impide el interbloqueo. (ROUTING=1).

Protocolo de Duato para mallas [6]: Este algoritmo usa el protocolo de Duato para obtener un algoritmo de encaminamiento completamente adaptativo. Se ha utilizado el algoritmo de orden de dimensión como algoritmo libre de bloqueos, añadiendo otros canales virtuales, los que se quieran, para la adaptabilidad completa. El algoritmo necesita al menos dos canales virtuales para su correcto funcionamiento. Si sólo se especifica un canal virtual el funcionamiento es exactamente el mismo que el de orden de dimensión. (ROUTING=2).

Completamente adaptativo puro para mallas: Este algoritmo utiliza cualquier canal de salida libre para enviar el paquete con un camino mínimo, sin tener en cuenta la posibilidad de que se produzcan bloqueos mortales. Naturalmente este algoritmo no está libre de interbloqueos, pero puede resultar interesante para comparar un algoritmo completamente adaptativo puro, sin restricciones, con otros algoritmos completa o parcialmente adaptativos libres de bloqueos que sí que pueden presentar alguna restricción. (ROUTING=3).

Todos estos algoritmos, incluyendo el algoritmo completamente adaptativo puro, asumen el supuesto de camino mínimo. Ejemplos de estos algoritmos se adjuntan en el código fuente, facilitando de esta manera cualquier cambio que se realice para implementar nuevos algoritmos.

En la sección dedicada a la interfaz gráfica se muestran todas las características del simulador relacionadas con los aspectos visuales, interactivos, ejecución en segundo plano, etc.

III. Arquitectura del simulador

La utilización de lenguajes orientados a objeto para el modelado hardware tiene varias ventajas: cada componente hardware puede especificarse de forma sencilla mediante una clase, y su comportamiento, junto con la interacción con otros elementos, puede modelarse con los métodos de la clase.

SimuRed utiliza esta idea para dividir el simulador completo en dos bloques: el núcleo y la interfaz de usuario. El núcleo del simulador es un conjunto de clases y métodos que definen una red de interconexión genérica y su comportamiento, junto con otros métodos que permiten a otras capas del programa enviar paquetes y avanzar el tiempo de simulación. El interfaz de usuario o *Front-End* es la capa de programa por encima del núcleo: funciona como interfaz entre el núcleo y el usuario, suministrando posibilidades de representación gráfica para mostrar la evolución de la simulación y los resultados. Simultáneamente permite una sencilla entrada de datos y especificación de la red y los parámetros de la simulación.

Es sencillo diferenciar estos dos bloques ya que por un lado toda la especificación del núcleo se localiza en los ficheros `red.h` y `red.cpp`, mientras que por el otro lado el código de la interfaz de usuario se encuentra típicamente en unos ficheros con el nombre de la aplicación (en este caso `simured.cpp` y `simured.h`). El núcleo del simulador se ha diseñado para que sea portable entre aplicaciones y sistemas operativos. Se ha realizado la compilación con `gcc` bajo Windows o Linux y también con el compilador de C++ de Borland.

En este momento se dispone de dos interfaces de usuario diferentes: una gráfica y más sofisticada que se ha escrito en Borland C++ Builder, y la otra mucho más sencilla que utiliza el modo línea de comandos. Esta última tiene la misma funcionalidad, salvo las facilidades gráficas, pero presenta un código mucho más sencillo de entender. De hecho, sirve como ejemplo de integración del núcleo del simulador en

cualquier otra herramienta.

A. El núcleo

El núcleo del simulador está formado por las clases y métodos que definen la estructura y comportamiento de la red. A partir de este núcleo resulta sencillo construir una interfaz de usuario o integrar el simulador como parte de otra aplicación. La estructura de clases se puede ver en la figura 2.

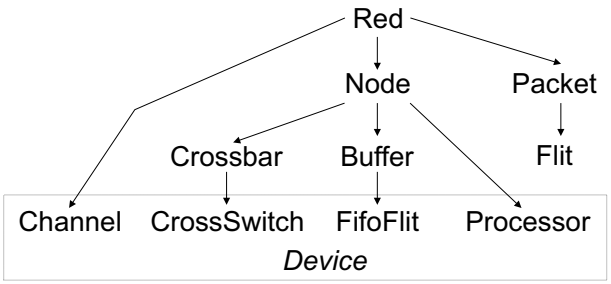

Fig. 2. Estructura jerárquica y clases del simulador.

La clase principal se llama *Red* y contiene todos los métodos y resto de clases. De esta clase principal cuelgan las clases *Channel*, *Node* y *Packet*. La clase *Channel* define la red de manera que todos los canales de la red son objetos de esa clase. Las clases *Node* y *Packet* definen los nodos y paquetes en la red y se describen a continuación.

La clase *Node* sirve para modelar cada nodo de la red. Consta a su vez de tres clases que son *CrossBar*, *Buffer* y *Processor*. La clase *CrossBar* modela la barra cruzada y tiene todavía una clase por debajo que es la *CrossSwitch*, que define cada uno de los conmutadores de salida de la barra cruzada. La clase *Buffer* modela las colas de entrada y salida y tiene una clase por debajo (*FifoFlit*) que sirve para modelar cada uno de los elementos de memoria (Flit) de la cola. Por último, la clase *Processor* define el comportamiento del procesador conectado al nodo.

La clase *Packet* sirve para modelar los paquetes en la red. Esta clase tiene a su vez la clase *Flit* que modela cada flit dentro del paquete.

Se ha creado una clase *Device* para empaquetar el comportamiento común de los elementos definidos por las clases *Channel*, *CrossSwitch*, *FifoFlit* y *Processor*. La característica común es que cada flit del paquete se encuentra siempre en alguno de estos *dispositivos* que son los elementos hardware básicos de la red.

Para generar la red únicamente es necesario crear un objeto de la clase *Red* con los parámetros de configuración necesarios (número de nodos, dimensiones, etc.). Una vez creada la red podemos inyectar paquetes simplemente creando un objeto de la clase *Packet* al cual se le puede pasar la longitud, origen, destino, etc. Por último, el usuario sólo necesita el método *RunCycle()* de la clase *Red* para realizar la simulación. Este método mueve los paquetes por la red y genera los cambios necesarios en las variables estadísticas suponiendo que ha pasado un ciclo de reloj. Resumiendo, inicialmente se crea la red y posteriormente se inyectan paquetes y se hace pasar el reloj ciclo a ciclo. Finalmente se pueden mostrar los resultados de los contadores estadísticos, generar nuevas estadísticas, etc. Incluso se puede mostrar esta información cada ciclo de reloj.

B. Simulación

La simulación está basada en eventos discretos. Los resultados mostrados corresponden a las medidas obtenidas de los paquetes moviéndose literalmente por la red, es decir, no se realiza ninguna estimación probabilística de lo que debería obtenerse, sino que en cada ejecución se obtiene la medida de lo que ocurre en la red modelada. Esto significa que es necesario inyectar un gran número de paquetes en la red para obtener unos resultados fiables.

La simulación se realiza a cada ciclo de reloj. Desde el punto de vista del usuario, lo único que se necesita conocer es que existe un método de la red llamado *RunCycle*; el resto de métodos no es imprescindible conocerlos pues son internos a la simulación, pero es interesante saber de ellos para ver cómo se implementado el simulador a nivel interno.

En el método *RunCycle* de la clase *Red* se realizan todos los cambios que tienen lugar en la red en un ciclo de reloj. Lo primero que se hace es mover los paquetes por la red. Para esto se recorre la lista de paquetes activos en ese instante y se ejecuta el método *RunCycle* de cada uno de los paquetes (se llama igual que el de la red global). Este método de cada paquete lo que hace es mover cada uno de los flits del paquete empezando por la cabecera. Este movimiento de flits por la red es la base de la simulación. Desde esta rutina es precisamente desde donde se llama a los algoritmos de encaminamiento. Si se quiere cambiar el algoritmo de encaminamiento de la red, basta con copiar la estructura de alguno de los que ya existen, crear una nueva función e incorporarla a este método (hay una sentencia `case` de C++ preparada para esto).

Las interfaces de usuario que se incluyen en el programa, tanto en su versión gráfica como no gráfica, permiten especificar simulaciones complejas para obtener conjuntos de gráficas en los que cambian varios parámetros. También incluyen la posibilidad de generar los paquetes a partir de ficheros de trazas o de forma aleatoria siguiendo una distribución uniforme a partir de una productividad dada.

C. Interfaz gráfica

Una de las características más interesantes del simulador, especialmente desde el punto de vista educacional, es la posibilidad de observar el estado de la red en cualquier instante y fijarse en el movimiento de los paquetes. La figura 3 muestra la ventana principal del simulador. En este caso se muestran las opciones de la pestaña dedicada a la simulación; en el resto de pestañas se pueden cambiar los diferentes parámetros de la red, ver gráficas, etc. En esta misma figura aparece superpuesta la ventana que muestra la red. Una porción de esta ventana se mostraba ya en la figura 1.

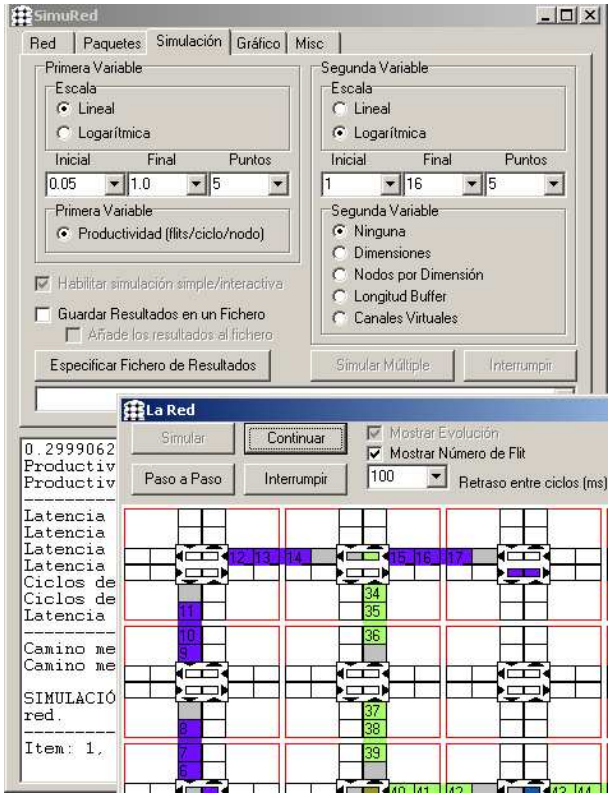

Fig. 3. Ventana principal del simulador.

Adicionalmente el interfaz visual permite generar gráficas a partir de los resultados de la simulación. La figura 4 muestra la ventana de gráficas de la herramienta donde se puede observar el resultado de una simulación. En esta gráfica aparece la latencia media de la cabecera del paquete para el algoritmo de orden de dimensión, aplicado a una malla 4x4, en función del tráfico de paquetes (productividad expresada en flits por nodo y por ciclo) para diferente número de canales virtuales (1, 2, 4, 8 y 16). Cada curva se corresponde con un número de canales virtuales y se representa utilizando un color diferente. Hay varios parámetros estadísticos que se pueden representar en las gráficas y cualquiera de ellos se puede poner tanto en el eje X como Y.

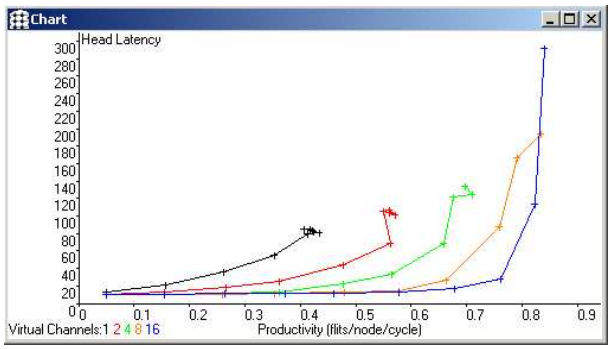

Fig. 4. Gráfica generada por el simulador donde se muestra la latencia de la cabecera en función de la productividad para diferente número de canales virtuales.

Esta interfaz de usuario gráfica funciona en Linux y Windows, si bien la versión de Linux puede fallar pues no es la plataforma bajo la que se ha desarrollado la herramienta. El programa está implementado tanto en inglés como en español y se puede conmutar entre estos idiomas desde el propio programa. La versión no gráfica en línea de comandos también se encuentra disponible en ambos idiomas.

IV. Conclusiones

Se ha presentado un simulador de redes de computadores visual e interactivo. Sus capacidades gráficas e interactivas lo hacen especialmente indicado para propósitos educacionales. Adicionalmente la especial arquitectura interna del simulador permite la integración de su núcleo en experimentos hechos a medida o en aplicaciones de investigación. Las fuentes del núcleo y los binarios listos para su funcionamiento son gratuitos y pueden descargarse de `http://simured.uv.es/`

Actualmente se trabaja en una versión paralela basada en Java.

V. Agradecimientos

Este desarrollo ha sido posible en parte gracias al proyecto TIC2001-3546 del Ministerio de Ciencia y Tecnología de España.

Referencias

[1] Jeffrey T. Draper and Jpydeep Ghosh, "Multipath E-Cube Algorithms (MECA) for Adaptive Wormhole Routing and Broadcasting in k-ary n-cubes," in *6th International Parallel Processing Symposium*, 1992, pp. 407–410.

[2] J Koller, J. Draper, and M Gorman, "ASNT Network Simulation Tools Survey and Requirements," Tech. Rep., USC Information Sciences Institute, 1995, http://www.isi.edu/asd/ams/private/asnt/docs/SimSpec.pdf.

[3] Jose M. García, Jose L. Sánchez, and P. González, "PE-PE: a Trace-Driven Simulator to Evaluate Reconfigurable Multicomputer Architectures," *Lectures Notes in Computer Science*, vol. 1184, pp. 302–311, 1996.

[4] Jennifer Rexford, Wu-Chang Feng, James Dolter, and Kang G. Shin, "PP-MESS-SIM: A Flexible and Extensible Simulator for Evaluating Multicomputer Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8–1, no. 1, pp. 25–40, Jan. 1997.

[5] Fernando Pardo, *SimuRed, manual del usuario*, Universidad de Valencia, 2004, http://simured.uv.es/.

[6] José Duato, Sudhakar Yalamanchili, and Lionil Ni, *Interconection Networks, an engineering approach*, IEEE Computer Society Press, EEUU, 1997.