UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Instalación, Configuración y Evaluación de la Red de Interconexión EXTOLL en un Entorno de Memoria Distribuida

Proyecto Final de Carrera

Ingeniería Informática

Autor: Javier Prades Gasulla

Director: Federico Silla Jiménez

14 de septiembre de 2014

Resumen

La computación de altas prestaciones o High Performance Computing (HPC) se inició con la aparición de los primeros supercomputadores, a finales de los años 60 y principios de los 70, y ha ido evolucionando hasta nuestros días, en los que está dominada por las arquitecturas High Performance Computing Clusters (HPCC). La red de interconexión es uno de los aspectos clave tanto en el desarrollo como en la implatación de estos sistemas cluster. Esta red de interconexión se encarga de proporcionar el soporte necesario para que los miles de nodos de estos clusters puedan intercambiar información entre sí de forma eficiente.

EXTOLL es una nueva arquitectura de red de interconexión, desarrollada por la Universidad de Heidelberg, que pretende establecerse en el sector del HPCC como una alternativa altamente eficiente y económica. EXTOLL proporciona mecanismos de comunicación eficaces tanto para mensajes de pequeño tamaño como para la transmisión de grandes cantidades de datos mediante mecanismos de Remote Direct Memory Access (RDMA). Cabe destacar que el diseño de EXTOLL, actualmente en fase de desarrollo, está implementado en una Field Programmable Gate Array (FPGA), lo que permite realizar modificaciones pero resta prestaciones. Se espera que, para abaratar costes y aumentar prestaciones, el diseño completo acabe integrándose en un chip Application-Specific Integrated Circuit (ASIC) en un futuro cercano.

Con el objetivo de evaluar el rendimiento de la nueva tecnología de interconexión EXTOLL en un entorno de memoria distribuida y compararlo con el obtenido usando una red de interconexión basada en Gigabit Ethernet, hemos configurado un prototipo de 64 nodos capaz de proporcionar 8TFLOPS de potencia pico. La red EXTOLL usada permite multiples topologías directas tanto en dos como en tres dimensiones. Sin embargo, si usamos Ethernet tendremos una única opción de conexión, formada por dos switches de 32 nodos conectados entre ellos. En este proyecto primero realizamos la caracterización de ambos sistemas de conexión basándonos en la latencia y el ancho de banda. Los resultados muestran un ancho de banda de 470 MB/s para EXTOLL y unos 100MB/s usando Ethernet. En cuanto a la latencia, al usar EXTOLL obtenemos una latencia cinco veces menor. Una vez realizada esta caracterización hemos evaluado el rendimiento de ambos sistemas utilizando

una serie de aplicaciones paralelas muy usadas en el campo de la dinámica molecular:

- Car-Parrinello Molecular Dynamics (CPMD)- Programa que contiene una implementación del método Car-Parrinello, diseñado para cálculos ab initio de dinámica molecular con la Density Functional Theory (DFT).
- DL-POLY- Es un paquete de subrutinas, programas y ficheros de datos, diseñado para facilitar las simulaciones de dinámica molecular de macromoléculas, polímeros, sistemas iónicos y soluciones.
- Large-scale Atomic Molecular Massively Parallel Simulator (LAMMPS)- Es un programa de dinámica molecular de los Laboratorios Nacionales de EEUU Sandia.
- GROningen MAchine for Chemical Simulations (GROMACS)- Es un programa de simulación de dinámica molecular que utiliza las ecuaciones newtonianas del movimiento para sistemas de centenares a millones de partículas.

Los resultados obtenidos muestran que el rendimiento usando EXTOLL es superior al de Gigabit Ethernet, obteniendo un speedup medio de 4x. Para finalizar la comparación de prestaciones, hemos adaptado una rutina paralela, capaz de generar imágenes pertenecientes al conjunto fractal de Mandelbrot, para poder apreciar el diferente nivel de prestaciones existente entre EXTOLL y Gigabit Ethernet en tiempo real. Esto es así porque la modificación realizada en esta aplicación genera muchas comunicaciones por tanto las diferencias de rendimiento de los sistemas de interconexión se ven reflejados en la velocidad de generación de las imágenes.

Finalmente hemos mostrado los diferentes mecanismos de control de flujo presentes actualmente en EXTOLL, ya que la motivación para desarrollarlos surgió de la realización de este trabajo, aunque su diseño y posterior implementación quedan fuera del alcance de este proyecto final de carrera.

Palabras clave: EXTOLL, Gigabit Ethernet, VELO, RMA, CPMD, DL_POLY, LAMMPS, GROMACS, Mandelbrot, control de flujo

Lista de Acrónimos

API Application Programming Interface.

ASIC Application-Specific Integrated Circuit.

ATU Adress Translation Unit.

BLAS Basic Linear Algebra Subprograms.

BTL Byte Transfer Unit.

CDC Control Data Corporation.

CPMD Car-Parrinello Molecular Dynamics.

CPU Central Processing Unit.

CRADA Cooperative Research and Development Agreement.

DDR2 Double Data Rate Type 2.

DFT Density Functional Theory.

EAM Embedded Atom Method.

FFTW Fastest Fourier Transform in the West.

FLOPS FLoating-point Operations Per Second.

FPGA Field Programmable Gate Array.

GNU GNU's Not Unix.

GROMACS GROningen MAchine for Chemical Simulations.

HOL Head-Of-Line blocking.

HPC High Performance Computing.

HPCC High Performance Computing Clusters.

HPL High Performance Linpack.

HT HyperTransport.

HTX HyperTransport eXpansion.

IBM International Business Machines Corporation.

INTEL INTegrated ELectronics.

ISC International Supercomputing Conference.

LAMMPS Large-scale Atomic Molecular Massively Parallel Simulator.

LAPACK Linear Algebra PACKage.

LiMIC Linux kernel Module for MPI Intra-node Communication.

MPI Message Passing Interface.

MTL Matching Transfer Layer.

MTU Maximum Transfer Unit.

NPT Constant Number (N), Pressure (P), and temperature (T); T is regulated.

NVT Constant Number (N), Volume (V), and temperature (T); T is regulated.

PCIe Peripheral Component Interconnect Express.

PIO Programmed Input/Output.

RDMA Remote Direct Memory Access.

RMA Remote Memory Access.

SC Supercomputing Conference.

SIMD Single Instruction, Multiple Data.

SPME Solid-Phase MicroExtraction.

UPC Unified Parallel C.

USB Universal Serial Bus.

VELO Virtualized Engine for Low Overhead.

VOQ Virtual Output Queuing.

Índice general

Capítulo 1

Introducción

La computación de altas prestaciones HPC es realizada en sistemas de supercomputación que proporcionan capacidades de cálculo muy superiores a los computadores de escritorio corrientes. Actualmente, y cada vez en mayor medida, la computación de altas prestaciones forma parte de nuestras vidas. Las prestaciones de los sistemas HPC se duplican cada año desde sus inicios. Esto significa un crecimiento x1000 aproximadamente cada 10-12 años, mayor incluso de lo que predice la ley de Moore.

El HPC se inició con la aparición de los primeros supercomputadores a finales de los 60 y principios de los 70, y ha ido evolucionando hasta nuestros días, en los que está dominado por las arquitecturas HPCC. La red de interconexión es uno de los aspectos clave tanto en el desarrollo como en la implatación de los sistemas HPCC. Esta red se encarga de proporcionar el soporte necesario para que los miles de nodos de los que están compuestos estos clusters puedan intercambiar información entre sí.

1.1. La computación de altas prestaciones

La computación de altas prestaciones HPC es realizada en sistemas de supercomputación que proporcionan capacidades de cálculo muy superiores a los computadores de escritorio corrientes. Actualmente, y cada vez en mayor medida, la computación de altas prestaciones forma parte de nuestras vidas, aunque la mayoría de la gente no es consciente, ya que es utilizada en muchos ámbitos de los que dependemos enormemente, tales como: la investigación necesaria para obtener la mayoría de los medicamentos que tomamos, la detección y tratamiento de nuestras enfermedades, el diseño de la mayoría de los vehículos que utilizamos, el cálculo de resistencias soportadas por la mayoría de las estructuras que forman nuestras ciudades, el análisis de riesgos financieros utilizado por las compañías aseguradoras, la predicción meteorológica, la defensa militar, la gestión de la red eléctrica, etc. Además, muchos de los dispositivos que usamos a diario como ordenadores personales, teléfonos móviles, tablets, etc... pueden ser considerados versiones reducidas de supercomputadores, ya que incorporan tecnología procedente del HPC, como el multi-core, las unidades de coma flotante y el procesamiento vectorial Single Instruction, Multiple Data (SIMD) y una memoria principal de alta velocidad.

1.2. Historia del HPC

El HPC se inició con la aparición de los primeros supercomputadores a finales de los 60 y principios de los 70. La compañía Control Data Corporation (CDC) desarrolló grandes trabajos de la mano de Seymour Cray, como el CDC 6600 y el CDC 7600 dominando el mercado por completo. Más tarde, a finales de los 70 y durante los 80, Seymour Cray siguió desarrollando supercomputadores como el Cray-1 o el Cray-2, ya desde su propia empresa, Cray Research Inc, dominando el mercado de nuevo. Durante la década de los 80 se instauran las máquinas vectoriales.

Figura 1.1: Consola del CDC 6600. Vía Wikimedia Commons

Figura 1.2: Cray-1. Vía Wikimedia Commons

Mientras que en los años 80 los supercomputadores utilizan tan solo unos pocos procesadores, en la década de los 90 empiezan a aparecer máquinas con miles de procesadores de "producción masiva" y es a partir de finales de esta década cuando empiezan a aparecer las primeras configuraciones en cluster basadas en microprocesadores comerciales. Estas configuraciones en cluster acaban por establecerse en la primera década de los años 2000 y son las que dominan el mercado de la computación de altas prestaciones actual. Están formadas por miles de nodos equipados con procesadores de propósito general baratos y una de sus principales virtudes es que estas configuraciones pueden ser fácilmente ampliadas, ya sea incrementando el número de nodos o las capacidades de éstos. La Figura 1.3 muestra una metáfora de la supremacía actual de los clusters en la computación de altas prestaciones.

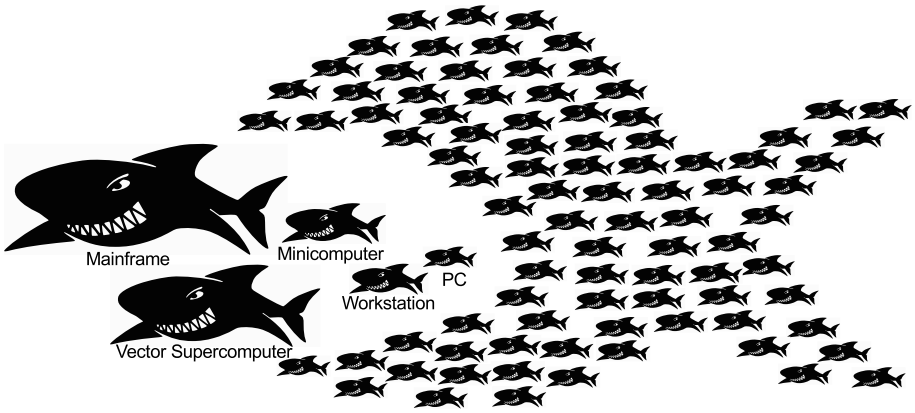

Figura 1.3: Símil que representa la evolución de las arquitecturas de los supercomputadores a lo largo de la historia

1.3. Evolución del HPC

El Top500 [12] desde 1993 realiza un ranking de los 500 computadores más rápidos del mundo, evaluando el rendimiento en TFLOPS que obtienen al ejecutar la aplicación High Performance Linpack (HPL). Este listado es actualizado dos veces al año, en junio (Europa) y noviembre (USA) coincidiendo con los congresos International Supercomputing Conference (ISC) y el Supercomputing Conference (SC) respectivamente. Por tanto podemos estudiar la evolución del HPC a través de la gran cantidad de información histórica proporcionada por el Top500. El único inconveniente es que tan solo tendremos la información de los computadores que envían sus resultados sin tener información de computadores privados o confidenciales.

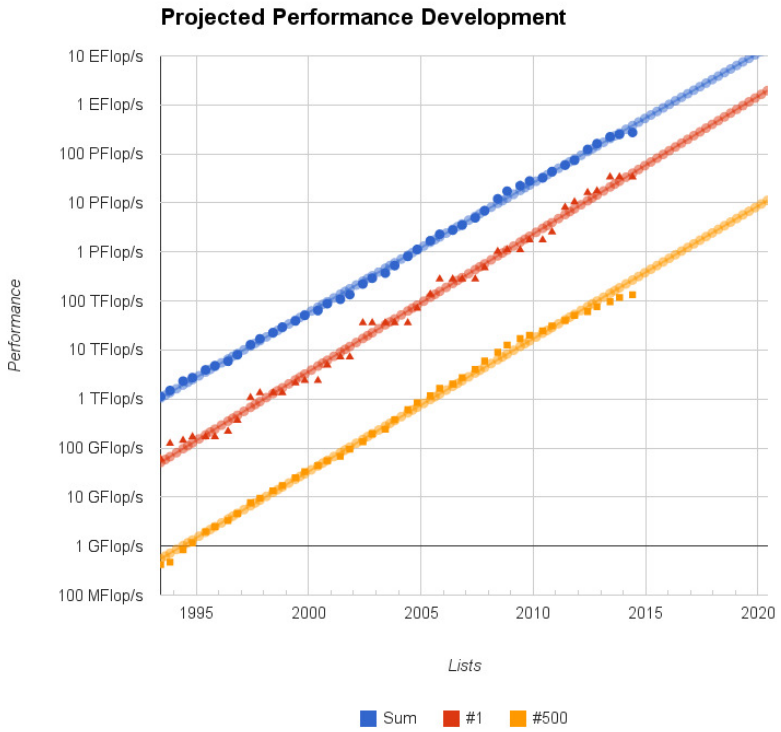

Figura 1.4: Tendencia seguida en el rendimiento de los sistemas HPC presentes en el Top500 (Fuente top500.org)

Si analizamos la Figura 1.4, que muestra los FLOPS alcanzados por el sistema más rápido del Top500, en rojo, del más lento, en amarillo, y de la

suma de los 500, en azul. Vemos cómo las prestaciones de los sistemas HPC se duplican cada año desde sus inicios, esto significa un crecimiento x1000 aproximadamente cada 10-12 años, mayor incluso de lo que predice la ley de Moore.

1.4. Las redes de interconexión

Tal como hemos visto en la Sección 1.2, la evolución histórica de las arquitecturas de computación de alto rendimiento ha ido desde supercomputadores únicos a los sistemas actuales formados por clusters de miles de nodos. Esta evolución también puede ser vista en las arquitecturas de los sistemas HPC presentes en el Top500 como muestra la Figura 1.5.

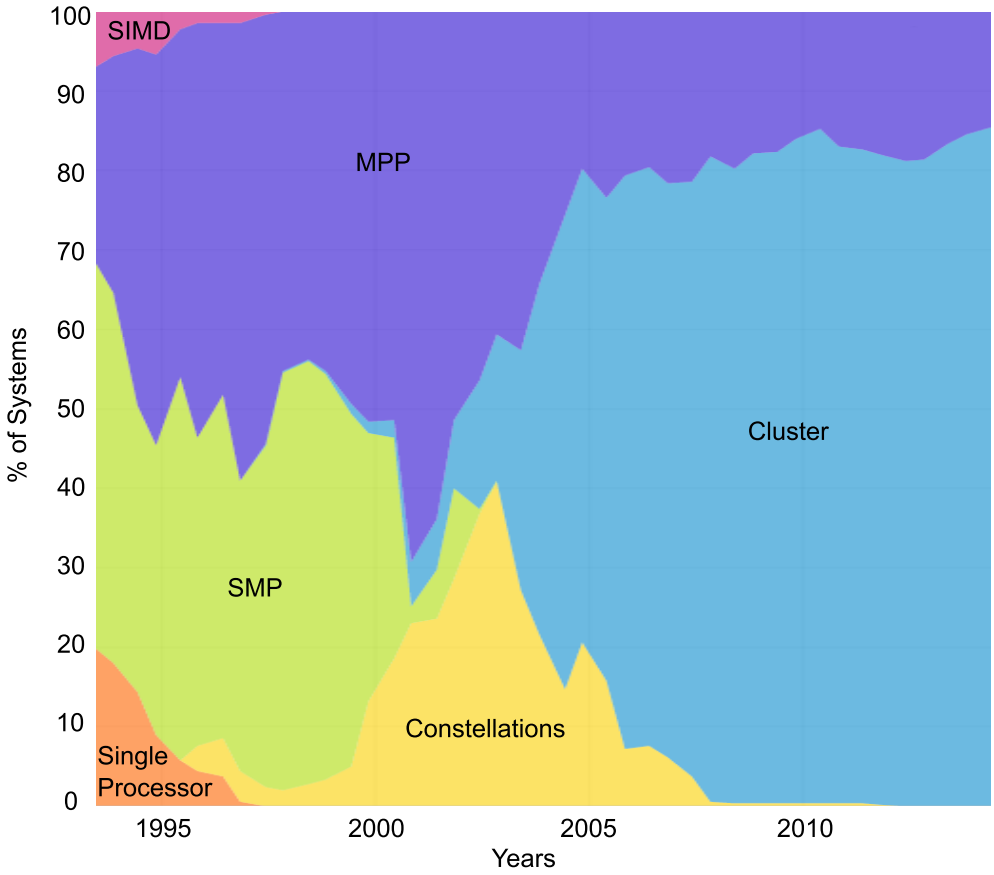

Figura 1.5: Evolución de las arquitecturas de los sistemas HPC presentes en el Top500 (Fuente top500.org)

Los clusters necesitan una tecnología de interconexión que permita que

todos sus nodos intercambien información de la forma más rápida y eficiente posible. Por tanto a medida que los clusters han ido ganando terreno a otros tipos de arquitectura, los sistemas de interconexión se han convertido en uno de los elementos más críticos e influyentes en las prestaciones alcanzadas por los mismos. Si analizamos la Figura 1.6 y la solapamos con la Figura 1.5 podemos observar exactamente que la implantación de los clusters de computación de alto rendimiento HPCC ha sido tanto la causa como el efecto del surgimiento de nuevos sistemas de interconexión mucho más rápidos que los anteriores y viceversa.

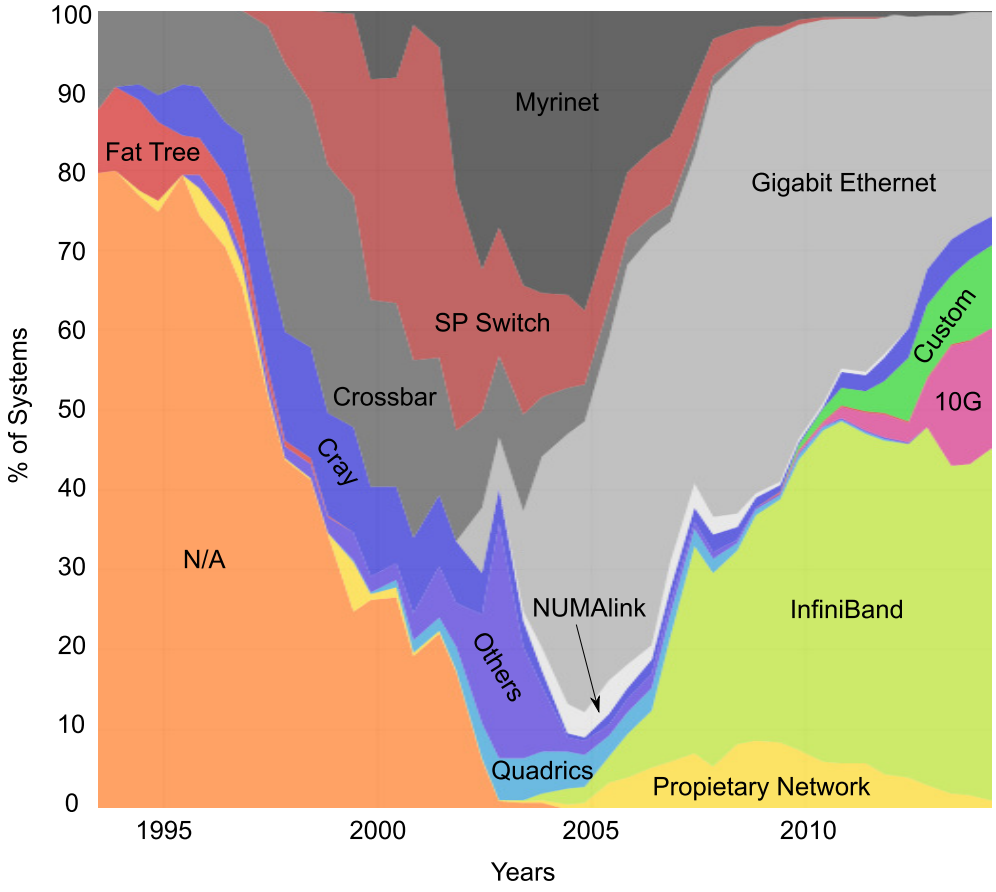

Figura 1.6: Evolución de las redes de interconexión de los sistemas HPC presentes en el Top500 (Fuente top500.org)

Capítulo 2

EXTOLL

EXTOLL es una nueva arquitectura de interconexión desarrollada por la Universidad de Heidelberg que permite interconectar un gran número de nodos de computación, con soporte dedicado para la transferencia eficiente tanto de pequeños mensajes como de grandes cantidades de datos al incluir mecanismos de RDMA. La red EXTOLL integra tanto el controlador del interfaz de red como el switch en un único circuito, lo que permite abaratar costes al no necesitar conmutadores externos. Este diseño está implementado actualmente en una FPGA montada en la tarjeta de red, la cual enlaza el nodo local con la red a través de HyperTransport (HT) o Peripheral Component Interconnect Express (PCIe). Además, el diseño incluye seis enlaces externos. El uso de estos seis enlaces permite la configuración de topologías tales como mallas o toros en dos y tres dimensiones. Se espera que, para abaratar costes y aumentar prestaciones, el diseño completo de la tecnología EXTOLL acabe integrándose en un único chip ASIC en un futuro cercano.

La tecnología de red EXTOLL optimiza las comunicaciones de mensajes cortos mediante Virtualized Engine for Low Overhead (VELO) mientras que para las transferencias de mayor tamaño utiliza Remote Memory Access (RMA), liberando a la CPU de las cargas de comunicación y permitiendo así un gran solapamiento entre cómputo y comunicaciones. Además utiliza Linux kernel Module for MPI Intra-node Communication (LiMIC) para realizar comunicaciones dentro de un mismo nodo.

Aunque internamente EXTOLL haga uso de VELO, RMA y LiMIC para las comunicaciones, existe un nivel mayor de abstracción para el programador, debido a que existen tanto una implementación de Message Passing Interface (MPI) como otra más cercana al modelo de memoria compartida basada en Unified Parallel C (UPC) sobre EXTOLL.

2.1. Desarrollo

EXTOLL es una nueva red de interconexión desarrollada por la Universidad de Heidelberg (Alemania) que pretende implantarse en el segmento del HPC, ya que permite interconectar un gran número de nodos de computación ofreciendo un excelentes prestaciones tanto en la transmisión de mensajes pequeños como en las transferencias de grandes volúmenes de datos. Además, permite abaratar costes al incluir el controlador del interfaz de red y el switch en un único circuito, de manera que los elementos de cableado sean los únicos recursos externos necesarios. El switch consta de seis puertos, permitiendo la configuración tanto de mallas como toros en dos y tres dimensiones a través de un cableado de fibra óptica. Todo este diseño está montado en la tarjeta de red, la cual enlaza el nodo local con la red a través de HT o PCIe.

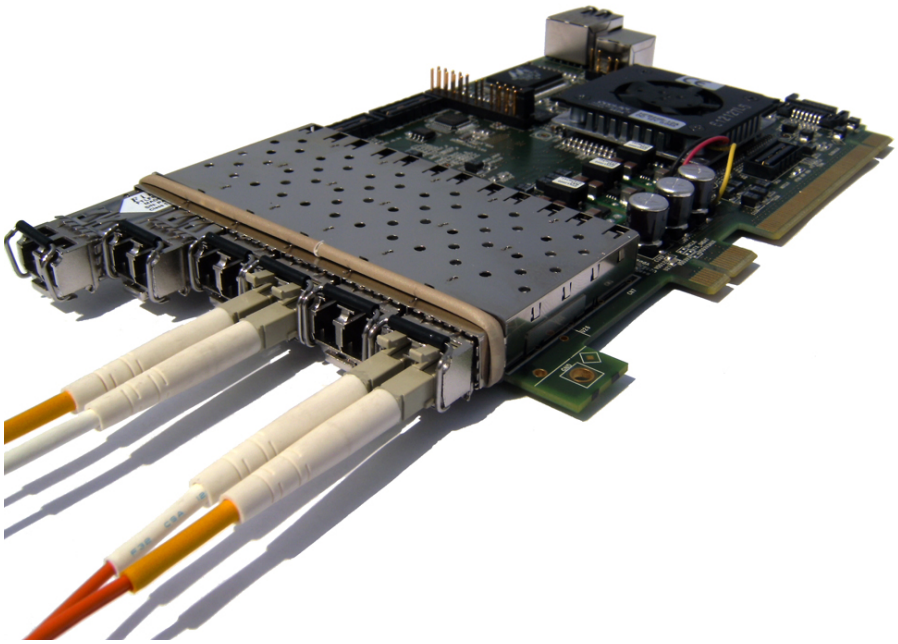

Figura 2.1: Tarjeta de red HTX, con transceivers y cables de fibra óptica

Actualmente EXTOLL se encuentra en fase de prototipo por lo que el diseño está implementado en una FPGA, lo que permite una mayor flexibilidad a la hora de realizar cambios pero limita el rendimiento. En un futuro cercano se espera que el diseño completo acabe integrándose en un chip ASIC aumentando las prestaciones.

2.2. Arquitectura

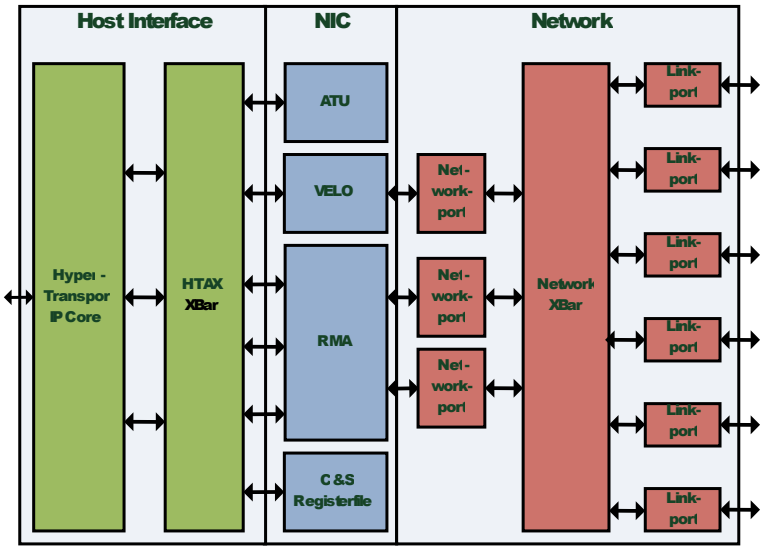

Figura 2.2: Diagrama de bloques de la arquitectura EXTOLL

La arquitectura de EXTOLL, mostrada en la Figura 2.2, está compuesta por una interfaz de host mostrada en el lado izquierdo (color verde), la cual está actualmente basada en HT[1], la interfaz de red, que incluye múltiples motores de comunicación, mostrados en el centro (color azul) y la sección de red con un switch y seis enlaces de red, mostrada en el lado derecho (color rojo).

Toda la arquitectura está implementada en un único chip. El switch integrado implementa una variante de Virtual Output Queuing (VOQ) para reducir el Head-Of-Line blocking (HOL) y emplea conmutación virtual cut-through que proporciona latencias de comunicación extremadamente bajas. Los interbloqueos son evitados mediante el uso de dos canales virtuales. El switch soporta también la entrega de paquetes en orden, lo que simplifica el diseño de protocolo software en las capas superiores. Una transmisión fiable está garantizada por un protocolo de retransmisión a nivel de enlace, característica que simplifica el diseño de los componentes software.

EXTOLL está diseñado para ser eficiente en esquemas de comunicación de grano fino, por lo que incluye un motor de comunicación especial llamado VELO que proporciona una transmisión optimizada para mensajes pequeños. Su interfaz optimizada entre hardware y software minimiza la sobrecarga

[1]Otras variantes de EXTOLL reemplazan la interfaz HT por una PCIe. Aunque desde el punto de vista de la arquitectura, no hay diferencia entre ambas implementaciones.

asociada al envío y recepción de paquetes. Esto contribuye no sólo a una baja latencia, sino también a la consecución de altas tasas de envío de mensajes. Al enviar un mensaje, básicamente, sólo una copia Programmed Input/Output (PIO) a una dirección especial es necesaria. Metadatos como el destinatario o la longitud son codificados en la dirección, por lo que la carga útil de escritura PIO completa (típicamente 64 bytes) está disponible para un uso por parte del software. En el lado receptor, los paquetes se escriben directamente en la memoria principal usando un único buffer en anillo, llamado mailbox. En [18] puede encontrarse información mucho más detallada acerca de VELO.

Aparte de proporcionar soporte a la comunicación de grano fino, EXTOLL también proporciona un eficiente mecanismo para las grandes transferencias de información llamado RMA que ofrece al usuario la semántica de PUT/GET. Una unidad de traducción de direcciones Adress Translation Unit (ATU) basada en hardware permite una transferencia segura de datos. La unidad de RMA libera casi completamente de carga a la Central Processing Unit (CPU) durante la tarea de comunicación, por lo que se obtiene un gran solapamiento junto con una escasa sobrecarga. Las notificaciones pueden generarse tanto en el lado origen como en el destino con el propósito de indicar la finalización de una petición PUT/GET. Más detalles acerca de RMA y ATU son proporcionados en [19].

La combinación de ambos motores de comunicación proporciona soporte eficiente para realizar envíos de cualquier tamaño de mensaje. A través del uso de las Application Programming Interface (API) libvelo y librma, el usuario o librería pueden decidir qué método de transferencia es más adecuado para un tamaño determinado de mensaje. Además, todos los componentes software están libres de cualquier tipo de sobrecarga asociada a la entrega en orden de los mensajes o la retransmisión por pérdida de paquetes.

2.3. MPI sobre EXTOLL

EXTOLL ofrece soporte software para ser usado mediante MPI [7] o UPC [1] a través de GASNet [14], [17]. Dado que en este trabajo únicamente utilizamos MPI para evaluar el rendimiento de EXTOLL, en este apartado únicamente se expondrán los principales puntos de la implementación MPI sobre EXTOLL.

El soporte MPI de EXTOLL está basado en la popular implementación de código abierto OpenMPI [10]. En el nivel inferior de MPI, las API libvelo y librma proporcionan acceso directo a nivel de usuario a la funcionalidad de los respectivos motores de comunicación dentro de EXTOLL. OpenMPI es una implementación de MPI altamente modular y orientada a componen-

tes. Existen distintas posibilidades para dar soporte a los distintos tipos de envío punto a punto, como son Byte Transfer Unit (BTL) o Matching Transfer Layer (MTL). EXTOLL utiliza MTL. MTL controla todo el manejo del protocolo, que incluye la realización tanto de envíos como de recepciones siguiendo la especificación MPI.

Tal y como se ha visto en la sección anterior, los mensajes pequeños hasta un determinado umbral se envían utilizando un motor llamado VELO. Por encima de este valor umbral los mensajes se envían usando RMA. El valor óptimo del umbral en el caso de MPI está establecido en 2KB, por lo que mensajes de tamaño igual o inferior a 2KB son enviados mediante VELO y el resto de mensajes son enviados utilizando RMA. Además, si la comunicación se realiza entre dos procesos MPI que están ejecutándose dentro de un mismo nodo. EXTOLL utilizará un nuevo mecanismo de comunicación basado en LiMIC [16].

2.3.1. VELO

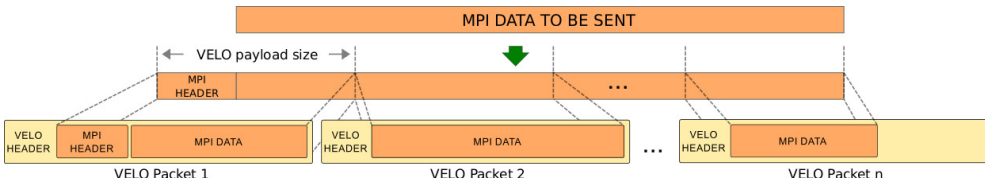

Figura 2.3: Transmisión típica de MPI bajo EXTOLL, usando el motor de comunicación VELO

El envío mediante VELO es realizado utilizando paquetes de tamaño 64 bytes. Al realizar envíos MPI mediante VELO, si la información a enviar no cabe en un sólo paquete VELO, se utilizarán tantos paquetes VELO como sea necesario. En el lado receptor se ensamblarán los diferentes paquetes VELO para recomponer el mensaje MPI si es necesario. La cabecera del mensaje MPI se incluirá en el primer paquete VELO. La Figura 2.3 muestra un envío típico de MPI usando múltiples paquetes VELO.

2.3.2. RMA

Los envíos de un tamaño superior al umbral de 2KB son enviados utilizando un protocolo de *rendez-vous* aprovechando RMA. En este protocolo, inicialmente se envía un paquete VELO que lleva la información que describe el buffer que en realidad va a ser enviado. Cuando el receptor recibe este paquete VELO realiza una o más operaciones GET mediante RMA. La unidad

de transferencia RMA son 4KB. Estas operaciones RMA son completadas sin necesidad de copias intermedias, produciendo de esta forma un protocolo muy eficiente.

2.3.3. LiMIC

Un tercer protocolo se ha aplicado para la comunicación dentro de un mismo nodo. Los procesos MPI que se encuentran en un mismo nodo pueden aprovecharse del uso de memoria compartida presente en los modernos nodos de múltiples procesadores y comunicarse a través de LiMIC.

Capítulo 3

Prototipo de 64 nodos y 1024 cores

Con el objetivo de evaluar la nueva tecnología de interconexión EXTOLL en un entorno de memoria distribuida se ha configurado un prototipo de 64 nodos capaz de proporcionar 8TFLOPS de potencia pico y con un consumo total de 64KW.

Nuestro prototipo presenta tanto una red de interconexión EXTOLL como otra basada en Gigabit Ethernet. Mediante el uso de EXTOLL podremos configurar topologías directas en dos y tres dimensiones. En dos dimensiones podremos configurar cualquier malla de tamaño igual o inferior a 8x8, además también podremos configurar un toro de dimensiones 8x8. Si decidimos utilizar topologías en tres dimensiones, podremos utilizar cualquier malla de tamaño igual o inferior a 4x4x4. Si utilizamos Ethernet únicamente tendremos una opción de conexión, formada dos switches, cada uno de ellos agregando 32 nodos y conectados ambos switches entre sí.

En el presente proyecto se ha partido del conjunto de 64 nodos, ya montados, y se ha trabajado en la instalación y configuración de la red de interconexión EXTOLL, creando los scripts necesarios para hacer una configuración (y verificación) automática de la topología de la red. En este capítulo mostramos, de forma simplificada, el desarrollo de los scripts de configuración de la topología. Se pueden encontrar más detalles al respecto en el Apéndice B.

3.1. Prototipo

Con el objetivo de evaluar la nueva tecnología de interconexión EXTOLL en un entorno de memoria distribuida se ha configurado un prototipo de 64 nodos que representa un total de 1024 núcleos de computación y 1TB de memoria principal, alcanzando 8TFLOPS de potencia pico y un consumo de 64KW. Cada nodo está equipado con una tarjeta EXTOLL y una tarjeta Ethernet.

3.1.1. Configuración de nodo

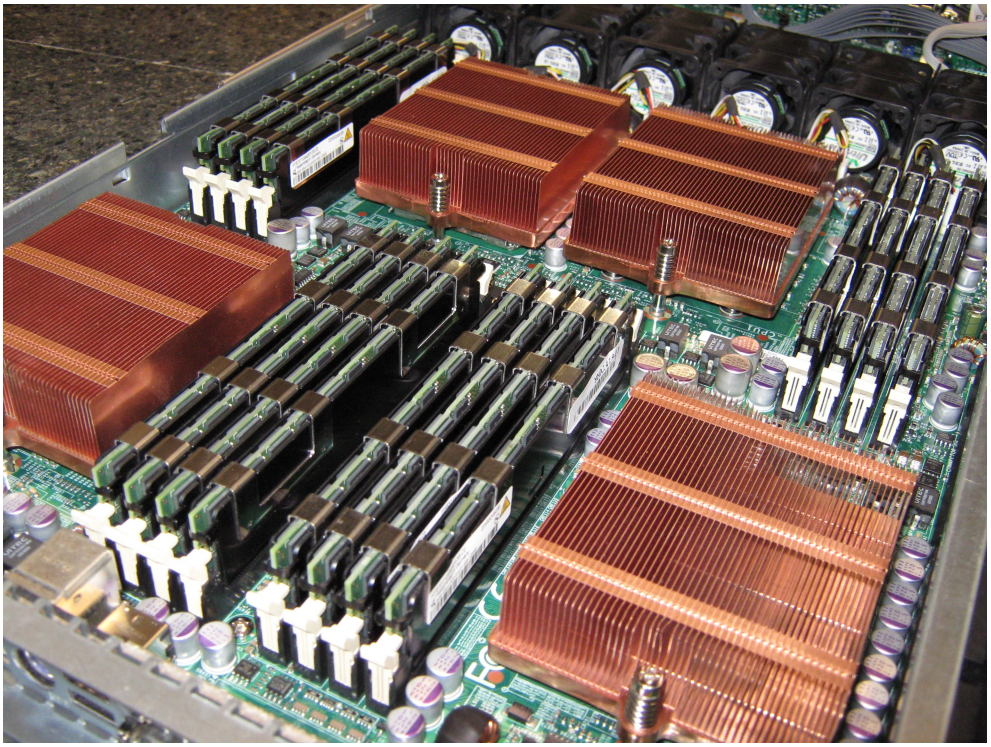

Figura 3.1: Interior de un nodo de nuestro prototipo

La Figura 3.1 muestra uno de los 64 nodos idénticos de los que se conforma nuestro prototipo. Cada nodo está equipado con la placa base de Supermicro H8QM8-2+ conteniendo cuatro procesadores Opteron de cuatro núcleos y una frecuencia máxima de 2.1GHz. Cada uno de estos procesadores dispone de 4GB a 800MHz de memoria Double Data Rate Type 2 (DDR2).

3.2. Interconexión EXTOLL

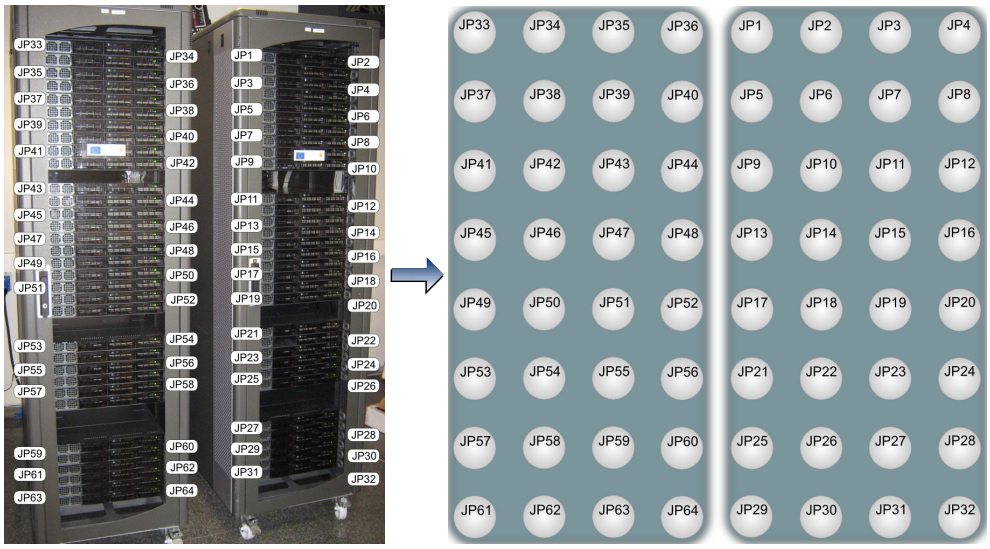

Figura 3.2: Distribución real a distribución esquemática de los nodos

El cableado de la red EXTOLL ha sido realizado pensando en dotar al prototipo de gran flexibilidad a la hora de utilizar mallas en dos y tres dimensiones. Es por ello que han sido utilizados todos los puertos de las tarjetas EXTOLL en la mayoría de los nodos para después elegir unos enlaces u otros a través de un simple script en bash, mostrado en el Apéndice B, según sean nuestras necesidades.

Para mostrar el proceso de cableado del prototipo utilizaremos una versión esquemática del mismo. La Figura 3.2 muestra el paso entre la distribución real de los nodos en el cluster, formada por dos racks de 32 nodos cada uno, a la distribución esquemática, formada por dos grupos de 4x8 nodos. Como puede deducirse de la Figura 3.2, cada uno de los grupos del diseño esquemático contiene los nodos de cada uno de los racks. Además, el identificador de cada nodo en el diseño esquemático coincide con el nombre del nodo al que representa. Estos identificadores están compuestos por la cadena "jp", la cual hace referencia al nombre del prototipo "jupiter", y un número comprendido entre 1 y 64.

3.2.1. Topologías en 2 dimensiones

El cableado realizado en el prototipo permite el uso de dos topologías diferentes en dos dimensiones, la malla y el toro. En este apartado mostra-

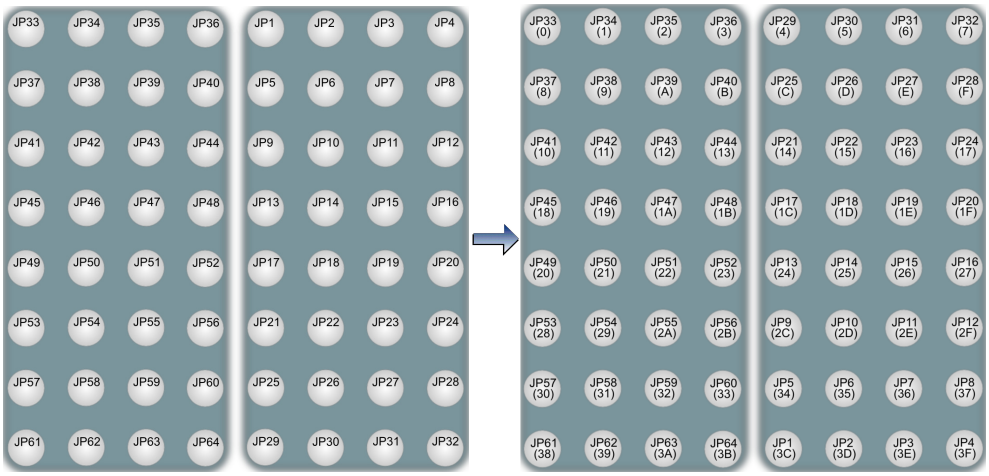

Figura 3.3: Inversión del grupo de la derecha y generación de los identificadores EXTOLL

remos la forma en la que se han cableado los nodos para permitirnos el uso de una u otra topología.

Malla 2D

La Figura 3.3 muestra un primer cambio realizado en el esquemático del cableado debido a la inversión de los nodos del grupo de la derecha, además también muestra junto a los nombres de cada nodo el identificador generado por EXTOLL para ese nodo en el caso de usar una malla 8x8. La inversión de los nodos del grupo de la derecha está motivada por la longitud de los cables de conexión, dado que el cableado entre un rack y otro es realizado por la parte superior del armario. Por lo tanto, al usar esta configuración todos los cables que cruzarán de un rack al otro tendrán la misma distancia. Si esta inversión de los nodos no se hubiera realizado nos veríamos obligados a utilizar desde cables muy cortos hasta cables excesivamente largos.

En la Figura 3.4 podemos apreciar tanto los puertos de enlace utilizados en cada uno de los nodos, como el cableado definitivo de la malla en dos dimensiones. Los puertos de enlace utilizados para el eje X serán por tanto el 0 y el 1, mientras que para el eje Y utilizaremos el 2 y el 3, quedando reservados los puertos 4 y 5 para el eje Z, tal y como podremos ver en el apartado de topologías en 3 dimensiones. Finalmente destacar que el sentido positivo en el eje X será de izquierda a derecha mientras que en el eje Y el sentido positivo será de arriba hacia abajo.

Por otra parte, por medio del script de configuración podremos crear cualquier malla 2D de tamaño inferior a 8x8 e incluso utilizar diferentes

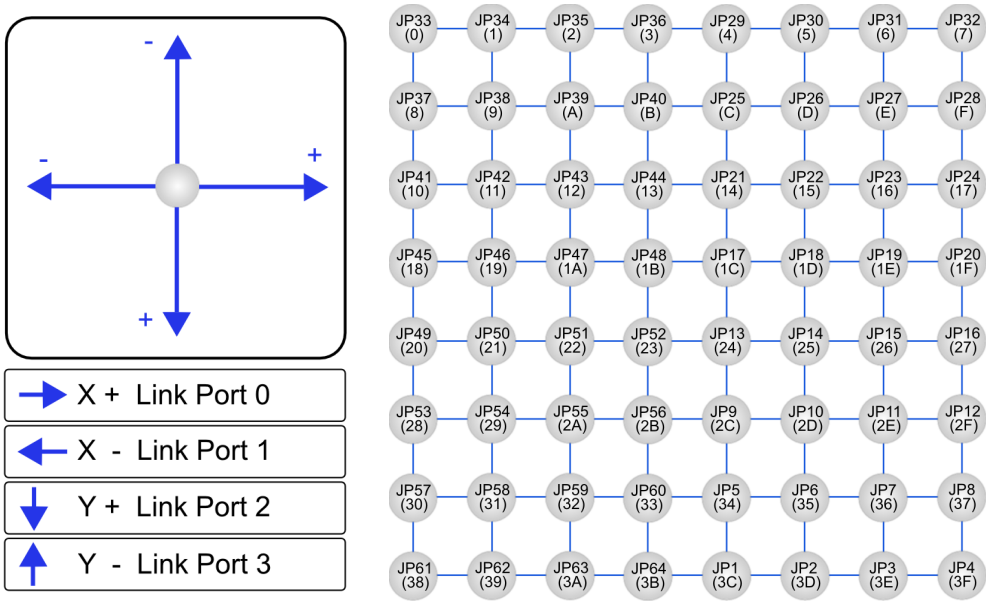

Figura 3.4: Malla 2D

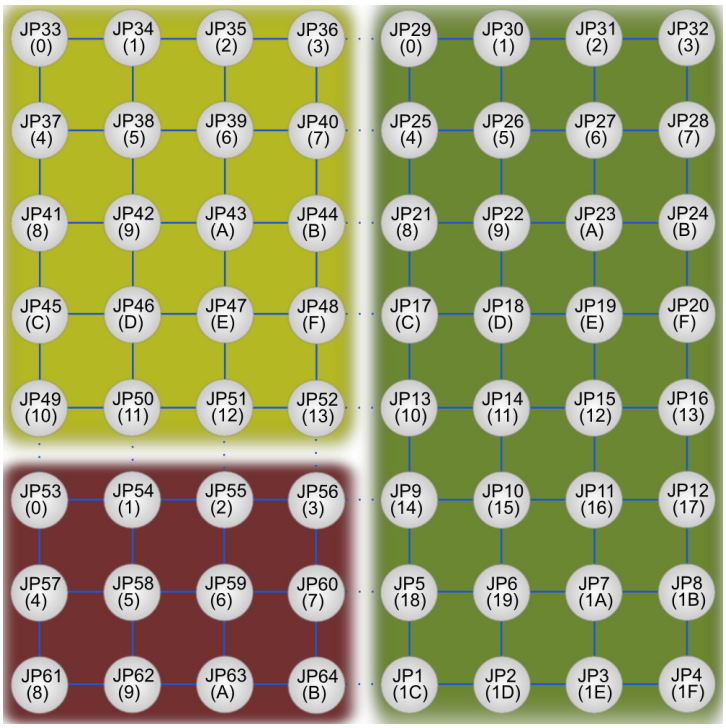

Figura 3.5: Posible configuración de mallas 2D independientes entre sí

mallas independientes a la vez, tal y como puede apreciarse en la Figura 3.5.

Toro 2D

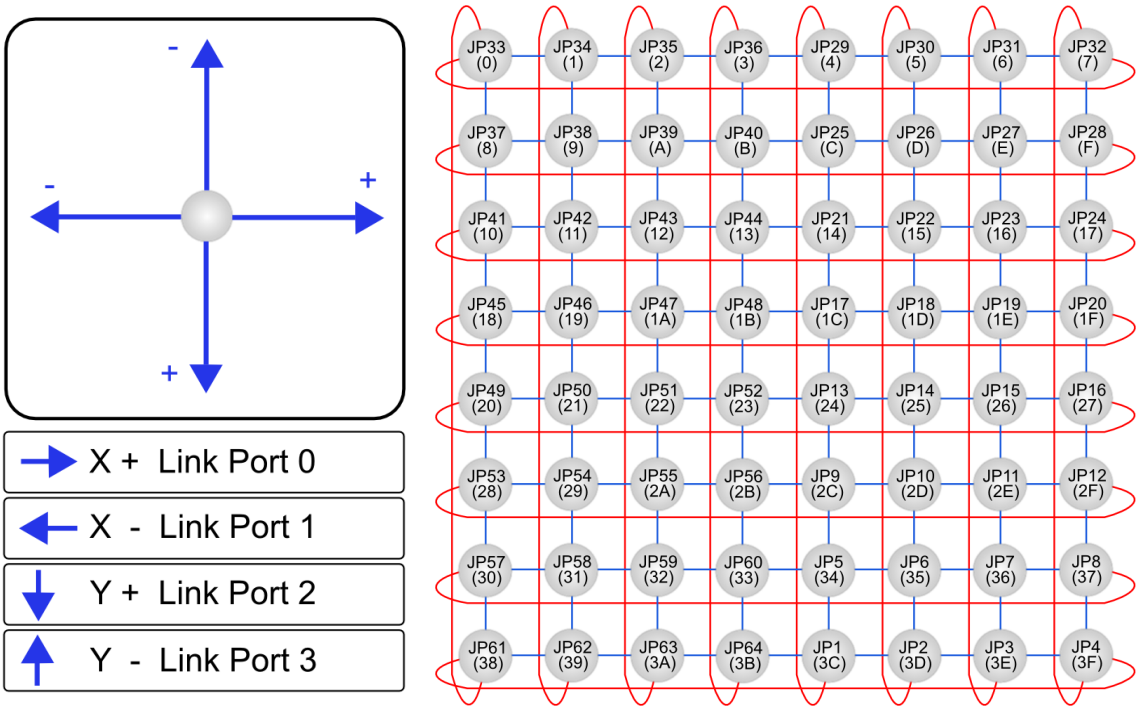

Figura 3.6: Toro 2D

Si creamos anillos tanto en el eje X como en el Y a partir de la configuración de la malla 2D vista anteriormente obtendremos la topología mostrada en la Figura 3.6, que corresponde a un toro 2D. Debemos tener en cuenta que al usar esta topología no podremos usar toros de otro tamaño, tal y como realizábamos con la malla 2D. Por tanto, si usamos la configuración de toro 2D tendremos un toro único de tamaño 8x8.

3.2.2. Topologías en 3 dimensiones

A parte de poder usar topologías en 2 dimensiones EXTOLL nos permite usar topologías en 3 dimensiones, ya que disponemos de hasta 6 puertos de enlace para poder crear topologías directas. Por tanto, usando los dos puertos de enlace sobrantes (4 y 5) podremos generar mallas en 3 dimensiones.

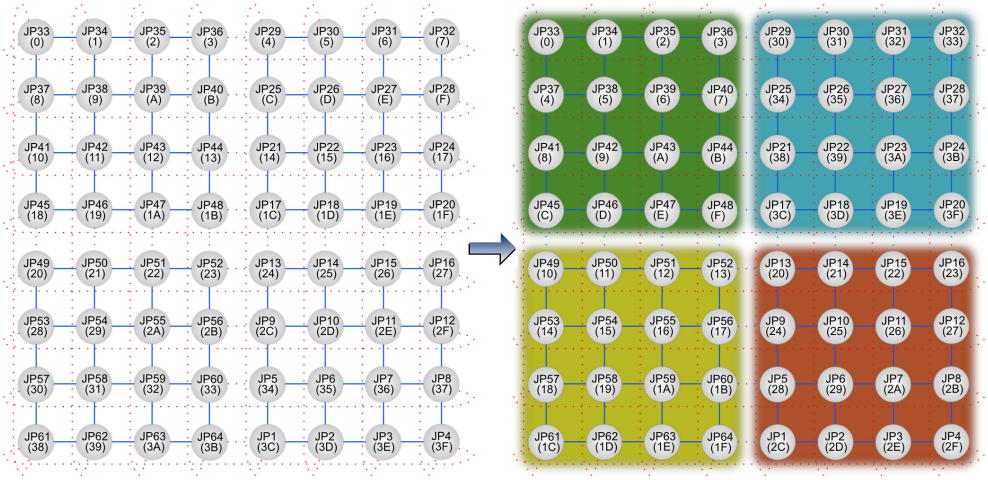

Figura 3.7: Paso de malla 2D a malla 3D

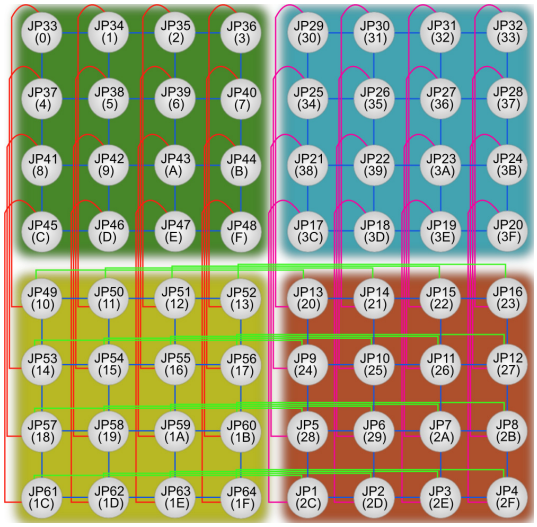

Figura 3.8: Enlaces completos malla 3D

Malla 3D

La Figura 3.7 nos muestra cómo desactivando el uso de algunos de los enlaces del toro 2D podemos crear cuatro planos X-Y de 4x4 nodos totalmente independientes. A partir de este punto podemos unir estos cuatro planos usando los puertos de enlace 4 y 5 a través de lo que será el eje Z, tal y como se aprecia en la Figura 3.8.

En la Figura 3.9 podemos apreciar tanto los puertos de enlace utilizados en cada uno de los nodos como el cableado definitivo de la malla en tres

dimensiones. Los puertos de enlace utilizados para el eje X serán por tanto el 0 y el 1, mientras que para el eje Y utilizaremos el 2 y el 3, reutilizando el cableado de la malla 2D. Los puertos 4 y 5 serán utilizados para el eje Z. Finalmente destacar que el sentido positivo en el eje X será de izquierda a derecha mientras que en el eje Y el sentido positivo será de arriba hacia abajo, de igual forma que la configuración 2D. El sentido positivo para el eje Z será hacia dentro del papel.

Además, por medio del script de configuración podremos crear cualquier malla 3D de tamaño inferior a 4x4x4 e incluso utilizar diferentes mallas independientes a la vez, tal y como puede apreciarse en la Figura 3.10.

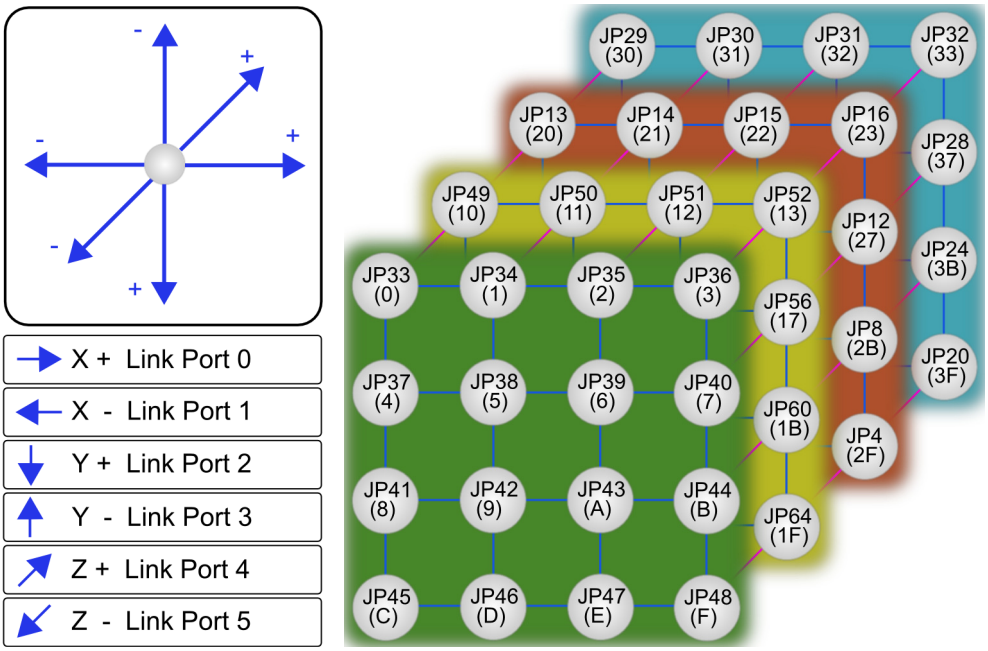

Figura 3.9: Malla 3D

3.3. Interconexión Ethernet

La red de interconexión Ethernet que presenta el prototipo pretende satisfacer las necesidades de gestión y administración. Se trata de una red Gigabit Ethernet y está formada por dos switches, uno por rack, a los que se conectan de forma independiente cada uno de los nodos del rack. Finalmente ambos switches están conectados por un cable. Este esquema de conexión resulta muy apropiado para las tareas de gestión y administración del prototipo pero presenta algunos puntos débiles si se utiliza para la comunicación

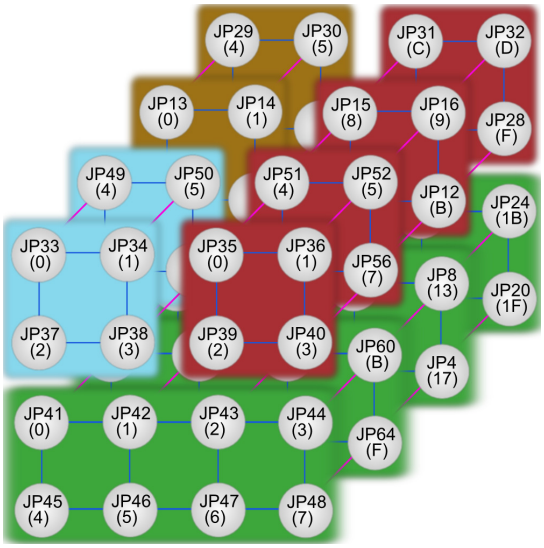

Figura 3.10: Ejemplo de configuración con diferentes mallas 3D independientes

de datos durante la ejecución de aplicaciones paralelas, tal y como se verá en el Capítulo 5.

Capítulo 4

Caracterización de EXTOLL y Ethernet

En este capítulo caracterizamos las redes de interconexión EXTOLL y Ethernet. El objetivo es, por tanto, obtener una serie de valores que nos ayuden a cuantificar el rendimiento real tanto de EXTOLL como de Ethernet y nos sirvan después para comparar estos sistemas de interconexión frente a otros disponibles actualmente. Para ello se medirá su comportamiento en términos de latencia y ancho de banda, ya que éstos dos valores describen el comportamiento de cualquier red de interconexión. También se determinará para EXTOLL el impacto que tiene en las prestaciones el tráfico existente en la red, tanto en otros nodos de la misma, como en el interfaz de red del nodo sobre el que se toman medidas.

El análisis se realiza por medio del uso de benchmarks escritos usando MPI. Esto introducirá un pequeño overhead respecto al uso de VELO y RMA sin ninguna capa software adicional, en el caso de EXTOLL. Sin embargo, nos permitirá una mayor flexibilidad y facilidad de programación a la hora de implementar las diferentes pruebas, además de poder realizar una comparación inmediata usando Ethernet.

4.1. Latencia

El propósito de esta primera prueba es medir la latencia de EXTOLL y Ethernet. Para ello hemos usado un test de PingPong presente en la popular suite de benchmarks para MPI de Intel [5]. Este test es realizado por dos procesos: el primer proceso inicia el PingPong por medio de un envío con un tamaño de mensaje de longitud conocida. Este mensaje es recibido por un segundo proceso que le responde al primero con otro envío de la misma longitud, finalizando la operación de PingPong. La medición del tiempo es realizada por el proceso que inicia la prueba, por lo que el tiempo obtenido, tiempo de ida y vuelta, debe ser dividido entre dos con el fin de calcular la latencia. Esta prueba ha sido realizada para diferentes tamaños de mensaje y a distinta distancia entre nodos.

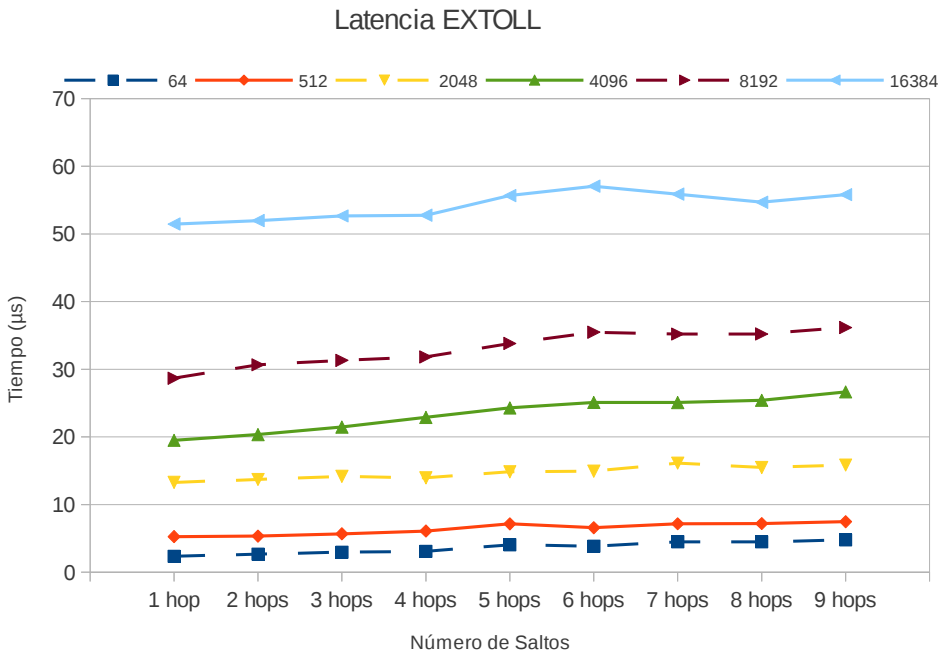

Figura 4.1: Latencia obtenida según tamaño de mensaje enviado y número de saltos para EXTOLL

Recordemos que EXTOLL utiliza dos métodos de transferencia diferentes. Cuando el tamaño del mensaje es menor o igual que 2048 bytes, EXTOLL utiliza VELO para realizar la transferencia, mientras que cuando el tamaño del paquete es mayor que 2048 bytes, EXTOLL utiliza RMA. El Maximum Transfer Unit (MTU) es diferente en cada caso. El tamaño de transferencia

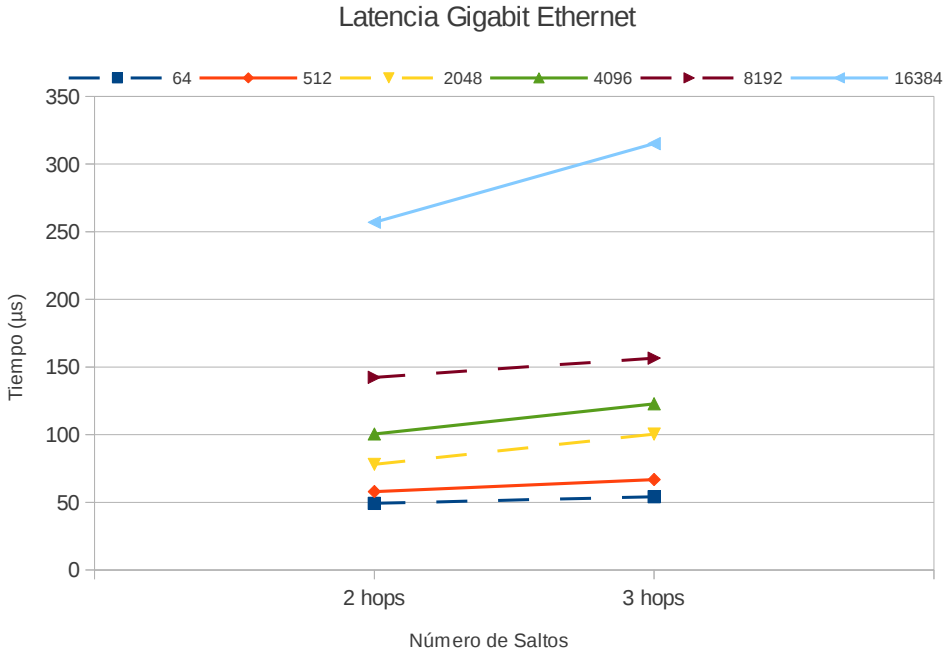

Figura 4.2: Latencia obtenida según tamaño de mensaje enviado y número de saltos para Gigabit Ethernet

máximo, MTU, de VELO es de 64 bytes, mientras que el MTU para RMA son 4096 bytes. Como puede verse en la Figura 4.1, la latencia de los mensajes depende en gran medida de la longitud de la transferencia y de la distancia, número de saltos, entre emisor y receptor. El aumento en la latencia debido a la distancia es lineal tanto cuando EXTOLL utiliza VELO como cuando utiliza RMA. Sin embargo, la pendiente o incremento de la latencia en cada salto es mayor cuando se utilizan transferencias RMA. En la Figura 4.2 puede apreciarse claramente cómo la latencia obtenida usando Ethernet es mucho mayor que la obtenida usando EXTOLL. Si comparamos el rango de tamaños donde en EXTOLL usamos VELO la latencia usando Ethernet es unas 10 veces mayor. En el rango de valores donde se usa RMA la latencia obtenida por Ethernet es unas 5 veces superior.

4.2. Ancho de Banda

El ancho de banda es una medida que nos ayuda a caracterizar cualquier sistema de comunicación ya que es un valor clave a la hora de elegir un sistema de interconexión u otro. En esta prueba enviamos una gran cantidad de

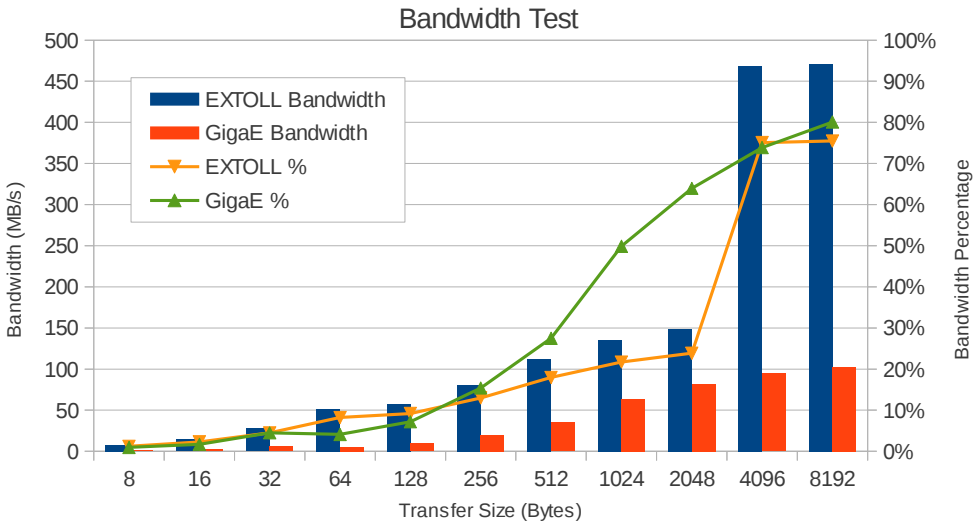

Figura 4.3: Ancho de Banda

mensajes entre dos nodos vecinos con el fín de calcular el ancho de banda máximo alcanzado por cada una de las redes de interconexión. Recordemos que el ancho de banda máximo teórico que puede alcanzarse mediante Gigabit Ethernet es de 125MB/s. Sin embargo para EXTOLL este límite máximo ronda los 624MB/s basándonos en trabajos como [15]. Por tanto esperamos obtener valores cercanos a estos límites teóricos. Para minimizar el tiempo perdido en la creación de mensajes, en lugar de usar el comando MPI_Send, usaremos el envío de 9999 mensajes asíncronos a través del comando MPI_Isend y luego sincronizaremos el proceso con un MPI_Send síncrono final.

La Figura 4.3 muestra el ancho de banda en MB/s y el porcentaje de ancho de banda alcanzado respecto a los valores teóricos, eje Y secundario. De acuerdo con estos resultados podemos establecer que, en el rango de transferencias que se realizan usando VELO en el caso de EXTOLL, el máximo ancho de banda alcanzado está alrededor de los 150MB/s con EXTOLL y 80MB/s con Ethernet. En el rango de la transferencias usando RMA el máximo ancho de banda alcanzado ronda los 470 MB/s o 75 % del valor teórico en el caso de EXTOLL y de 100MB/s o 80 % del total en el caso de Ethernet. En transferencias usando mensajes mayores de 8 KB el ancho de banda alcanzado se mantiene para ambos casos. En esta prueba podemos ver, en el caso de EXTOLL, de nuevo las diferencias debidas al MTU usado, porque cuando EXTOLL cambia el método de transferencia, entre VELO y RMA, el ancho de banda se incrementa notablemente. Si comparamos los resultados obtenidos mediante EXTOLL y Ethernet vemos cómo el ancho de banda

máximo alcanzado por EXTOLL es alrededor de 5 veces mayor que el de Ethernet. Sin embargo Ethernet presenta un incremento del ancho de banda respecto al tamaño de la transferencia mucho más suave que EXTOLL. En el caso de EXTOLL el ancho de banda presenta un cambio muy abrupto entre las zonas VELO y RMA.

4.3. Comportamiento de la tarjeta HTX de EXTOLL

En esta sección se evaluará el comportamiento de la tarjeta HyperTransport eXpansion (HTX) de EXTOLL, evaluando tanto la carga en la interfaz de red producida por los datos intercambiados como la carga producida por tráfico adicional.

4.3.1. Influencia de la carga en la interfaz de red

Otro aspecto importante a la hora de evaluar el rendimiento de EXTOLL en nuestro cluster es saber cómo la carga en la interfaz de red (tarjeta HTX), ya sea en el nodo emisor o en el nodo receptor, afecta a la latencia. Para analizar esta característica, en esta prueba vamos a observar el cambio en el rendimiento cuando aumentamos el número de procesos que realizan envíos o recepciones de forma simultánea en la misma máquina. Los nodos del clúster tienen 4 procesadores Quad-Core. Por lo tanto podemos ejecutar 16 procesos independientes y simultáneos en el mismo nodo. Esta prueba se compone de dos partes:

- **"Uno a Muchos"**: En esta parte evaluamos el cuello de botella que se produce en el nodo emisor al realizar múltiples envíos de mensajes desde un nodo a varios. El nodo emisor realizará uno, dos, cuatro, ocho o dieciseis envíos simultáneos que serán recibidos por uno, dos, cuatro, ocho o dieciseis nodos diferentes. En el caso más extremo un nodo realizará dieciseis envíos simultáneos a otros dieciseis nodos.

- **"Muchos a Uno"**: En esta parte evaluamos el cuello de botella que se produce en el nodo receptor al realizar múltiples envíos de mensajes desde varios nodos a uno solo. Se realizarán uno, dos, cuatro, ocho o dieciseis envíos simultáneos desde distintos nodos que serán recibidos por un solo nodo. En el caso más extremo un nodo recibirá dieciseis envíos simultáneos desde otros dieciseis nodos.

Tanto en la parte “Uno a Muchos” como en la parte “Muchos a Uno” el tiempo medido es el tiempo requerido para completar la operación por el proceso más lento. En la parte “Uno a Muchos” vamos a medir el tiempo de la operación de envío y en la parte “Muchos a Uno” vamos a medir el tiempo de la operación de recepción. Además, en las dos partes de esta prueba se ha utilizado una distribución justa de los nodos en el clúster que garantiza una carga homogénea en los enlaces involucrados. Por otra parte, hemos repetido esta prueba para varios tamaños de mensaje.

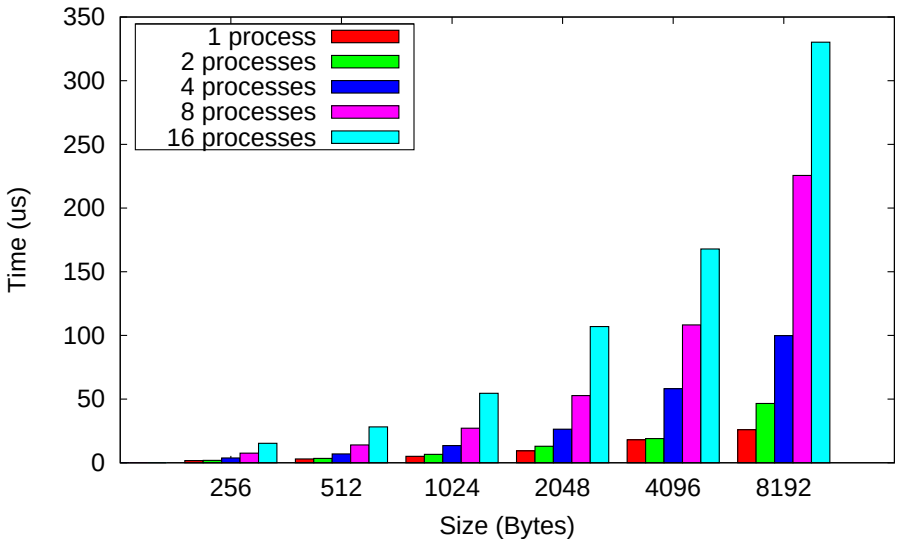

Figura 4.4: “Uno a Muchos”

Las Figuras 4.4 y 4.5 muestran los resultados de estas pruebas. Los resultados son similares. En ambas gráficas podemos ver claramente que, o bien aumentando el número de procesos en una máquina o por el aumento del tamaño de los mensajes utilizado, el tiempo total de la operación aumenta. La razón de esto es que cuando se aumenta el número de procesos en un solo nodo o cuando aumentamos el tamaño del mensaje, en realidad estamos aumentando el número de bytes procesados por la tarjeta HTX.

Si nos fijamos en los valores obtenidos observaremos un cambio de tendencia cuando el tamaño de mensaje es de 4096 bytes. Para tamaños más pequeños de mensaje, el tiempo requerido para completar la operación en cualquiera de las dos gráficas aumenta linealmente con el tamaño del mensaje. Por ejemplo, el tiempo requerido para mensajes de 2048 bytes es dos veces el tiempo requerido para los de 1024 bytes. Sin embargo, el tiempo requerido para mensajes de 4096 bytes no llega a ser el doble del tiempo requerido para los de tamaño igual a 2048 bytes. Este cambio de tendencia

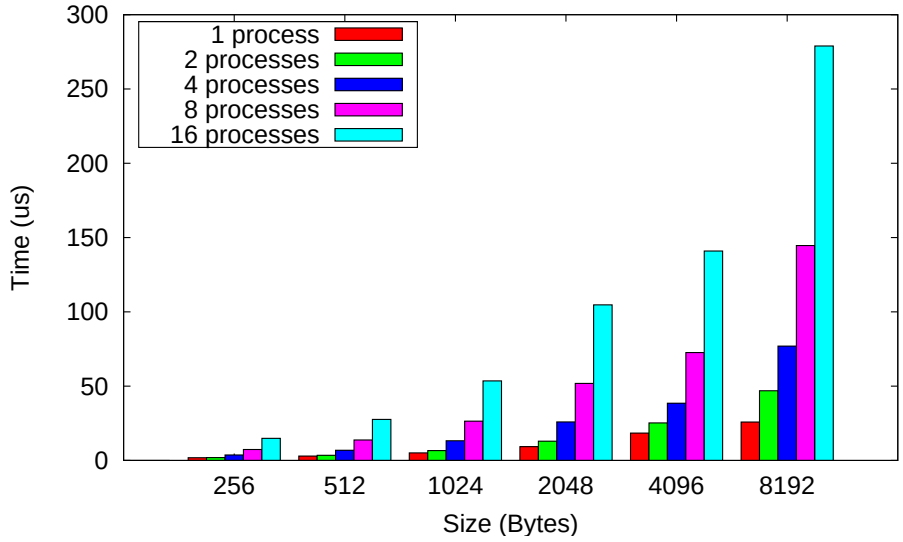

Figura 4.5: "Muchos a Uno"

también tiene que ver con la diferente MTU utilizada por VELO y RMA. Cuando EXTOLL utiliza RMA, el sistema alcanza un ancho de banda mayor que cuando EXTOLL utiliza VELO. Así que podemos ver un mejor funcionamiento de EXTOLL cuando el tamaño de transferencia es superior a 2048 bytes.

4.3.2. Influencia de presencia de tráfico en la red

Hasta ahora hemos caracterizado EXTOLL en términos de latencia y ancho de banda. Ademas hemos evaluado cómo afecta la carga de procesamiento en la interfaz de red. Sin embargo, todos estos aspectos pueden verse afectados si hay tráfico adicional en la red. Es decir, todas estas características de EXTOLL han sido analizadas en condiciones ideales, sin ningún otro tipo de actividad en la red que la estrictamente necesaria para la realización de las pruebas. En esta sección vamos a generar tráfico adicional que pueda alterar o colisionar con nuestras pruebas, y vamos a medir cómo afecta este tráfico a la latencia. Con el fín de obtener unos resultados más cercanos a un comportamiento real. Para ello realizaremos una transferencia principal de tipo PingPong con las modificaciones vistas en la Sección 4.2 y, sobre ella, analizaremos la influencia del tráfico. Esta transferencia principal se realizará entre los nodos identificados como 4 y 18 en la Figura 4.6.

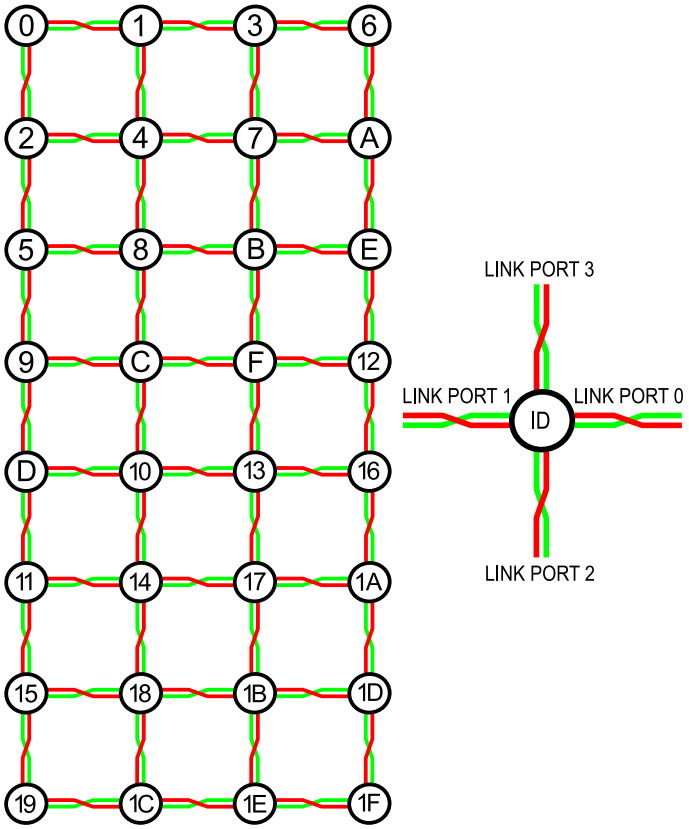

Figura 4.6: Configuración de la malla 2D EXTOLL utilizada para la caracterización.

Se realizarán tres pruebas diferentes para medir la influencia del tráfico:

- Tráfico adicional (eje X) que cruza la transferencia principal (eje Y). Para ello se generarán tramas de tráfico entre los nodos 2 y 7, 5 y B, 9 y F, D y 13, 11 y 17, 15 y 1B de la Figura 4.6. Además, vamos a utilizar mensajes de diferentes tamaños.

- Tráfico en la misma dirección que la transferencia principal (eje Y). Se generará tráfico entre el nodo identificado como 1 y el identificado como 1C en la Figura 4.6. En este caso también vamos a generar tráfico con diferente tamaño de mensaje.

- Tráfico uniforme (ejes X e Y), es decir en todas direcciones. Vamos a generar tráfico entre todos los nodos de la malla. En este caso los mensajes enviados por diferentes nodos también tendrán diferentes tamaños.

Como la transferencia en la que se observa la influencia de tráfico adicional se lleva a cabo enteramente en el eje Y, el tráfico en el eje X no debe

afectar demasiado. En la Figura 4.7 podemos observar que la latencia no varía con el tráfico adicional en el eje de abscisas. Sin embargo, cuando el tráfico adicional es en la misma dirección que la transferencia principal (eje Y), este tráfico interfiere directamente con dicha transferencia y la latencia aumenta significativamente respecto a la obtenida en ausencia de tráfico. En la Figura 4.8 puede apreciarse dicha situación.

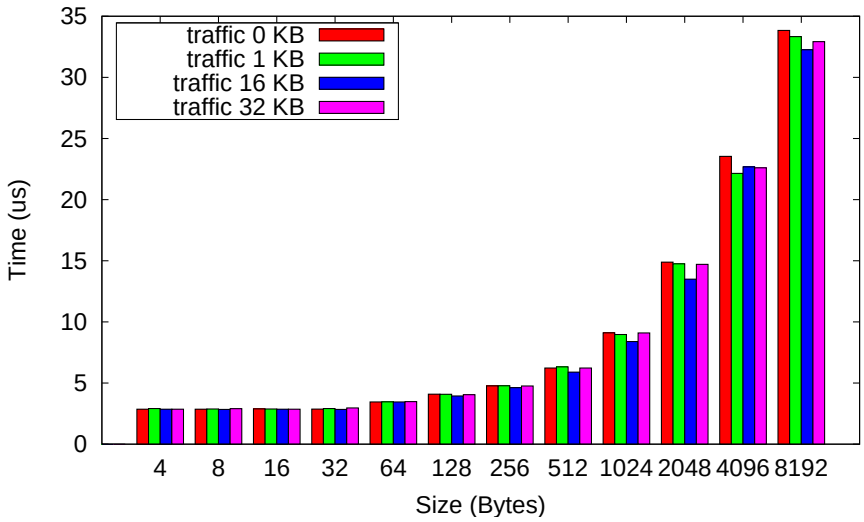

Figura 4.7: Tráfico adicional en el eje X

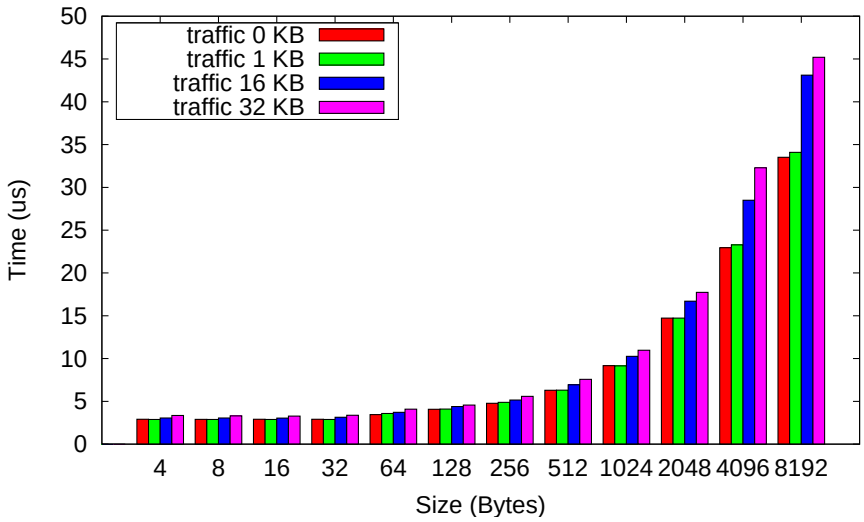

Figura 4.8: Tráfico adicional en el eje Y

Por tanto, cuando el tráfico estará distribuido uniformemente por toda la red, éste afectará también a la transferencia principal tal y como puede apreciarse en la Figura 4.9. Además cuanto mayor sea el tamaño de transferencia utilizado en la generación de tráfico adicional mayor será su efecto sobre la latencia. En el caso de tráfico uniforme, el tráfico inyectado a la red es mayor que en los otros dos casos anteriores. Por lo tanto la latencia es la más alta.

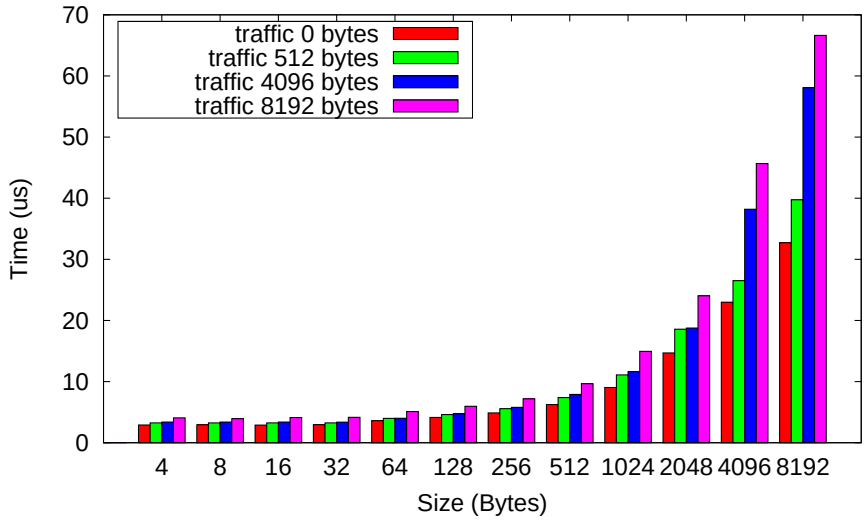

Figura 4.9: Tráfico Adicional Uniforme

Capítulo 5

Comparación de prestaciones mediante aplicaciones reales

En este capítulo se realiza una exhaustiva comparación del rendimiento de los sistemas de interconexión EXTOLL y GigaEthernet. Para ello han sido instaladas y configuradas una serie de aplicaciones muy utilizadas en el campo de la dinámica molecular. Las aplicaciones son:

- *CPMD- Programa que contiene una implementación del método Car-Parrinello, diseñado para cálculos ab initio de dinámica molecular con la DFT.*
- *DL-POLY- Es un paquete de subrutinas, programas y ficheros de datos, diseñado para facilitar las simulaciones de dinámica molecular de macromoléculas, polímeros, sistemas iónicos y soluciones.*
- *LAMMPS- Es un programa de dinámica molecular de los Laboratorios Nacionales de EEUU Sandia.*
- *GROMACS- Es un programa de simulación de dinámica molecular que utiliza las ecuaciones newtonianas del movimiento para sistemas de centenares a millones de partículas.*

Los resultados obtenidos muestran que el rendimiento usando EXTOLL es superior al de 1 GigaBit Ethernet obteniendo un SpeedUp medio de 4x.

5.1. Configuración de la interconexión

En este apartado explicamos brevemente la configuración de red, para EXTOLL y Ethernet, usada en las pruebas que se verán en los apartados siguientes. Para cada una de las pruebas de cada aplicación se usarán cinco posibles configuraciones de nodos:

- Configuración de 4 nodos - EXTOLL será configurado mediante una malla 2D 2x2. Ethernet utilizará 4 nodos conectados a un mismo switch.
- Configuración de 8 nodos - EXTOLL será configurado mediante una malla 3D 2x2x2. Ethernet utilizará 8 nodos conectados a un mismo switch.
- Configuración de 16 nodos - EXTOLL será configurado mediante una malla 2D 4x2x2. Ethernet utilizará 16 nodos conectados a un mismo switch.
- Configuración de 32 nodos - EXTOLL será configurado mediante una malla 3D 4x4x2. Ethernet utilizará 32 nodos conectados a un mismo switch (es decir, se utilizan todos los nodos pertenecientes a un mismo rack).
- Configuración de 64 nodos - EXTOLL será configurado mediante una malla 3D 4x4x4. Ethernet utilizará dos switches conectados entre sí. Cada uno de los switches albergará 32 nodos.

El número de procesos MPI por nodo será el mayor posible dependiendo de la escalabilidad que permita el test de cada una de las aplicaciones. Para la aplicación GROMACS únicamente utilizaremos un proceso por nodo, llegando hasta un máximo de 64 procesos. La aplicación CPMD nos permite una escalabilidad de hasta 128 procesos MPI por lo que ubicaremos dos procesos MPI por nodo. Finalmente, las aplicaciones DL_POLY y LAMMPS serán evaluadas usando el máximo de cores por nodo, 16 procesos MPI, llegando a un total de 1024 procesos MPI en el caso de 64 nodos.

5.2. CPMD

CPMD [11]. Contiene la implementación paralelizada, usando MPI, del método Car-Parrinello mediante onda plana/aplicación pseudopotencial de la DFT, especialmente diseñada para la dinámica molecular ab-initio.

El consorcio CPMD se estableció en 2001 con el fin de coordinar el desarrollo y distribución del código CPMD. Es una organización virtual que

comprende todos los usuarios y desarrolladores del código CPMD en todo el mundo. Esta organización está coordinada por el Dr. Alessandro Curioni (Gerente del Departamento de Computación Cognitiva y Ciencias Computacionales en el International Business Machines Corporation (IBM) Zurich Research Laboratory) y fué fundado por el Prof. Michele Parrinello y el Prof. Wanda Andreoni.

CPMD tiene derechos de autor conjuntamente por la Corporación IBM y el Instituto Max-Planck de Stuttgart.

5.2.1. Evaluación CPMD

Para evaluar el comportamiento de la aplicación CPMD hemos utilizado un conjunto de pruebas muy conocidas de dicha aplicación, CPMD Test Suite [3]. De entre todas las pruebas presentes en esta recopilación hemos ejecutado una simulación con 32 moléculas de agua, variando algunos de los parametros de entrada como son la opción de optimización al generar la función de onda y el número de grupos de trabajo. Como ya hemos dicho en el apartado anterior cada una de estas pruebas ha sido ejecutada con un total de 2 procesos MPI por nodo, alcanzando un total de 128 procesos MPI al usar los 64 nodos del prototipo.

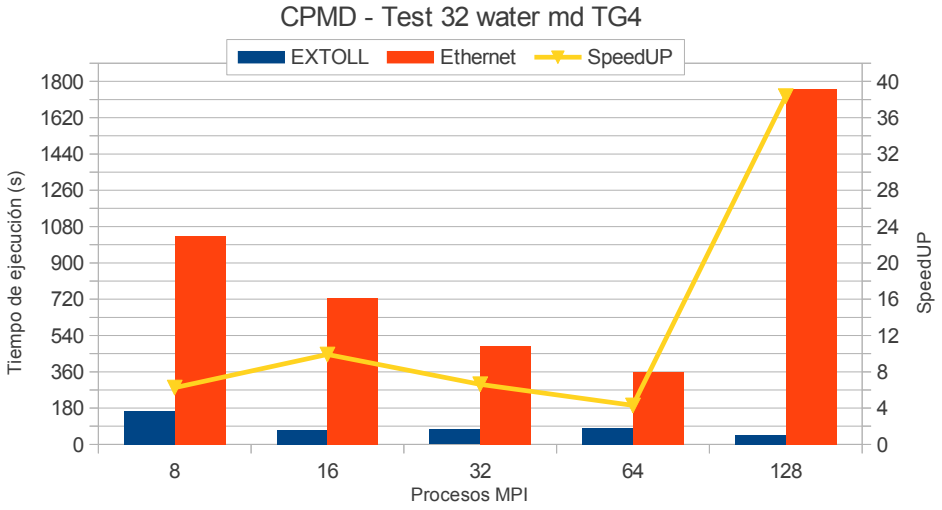

Figura 5.1: Simulación con 32 moleculas de agua sin optimización y 4 grupos de trabajo

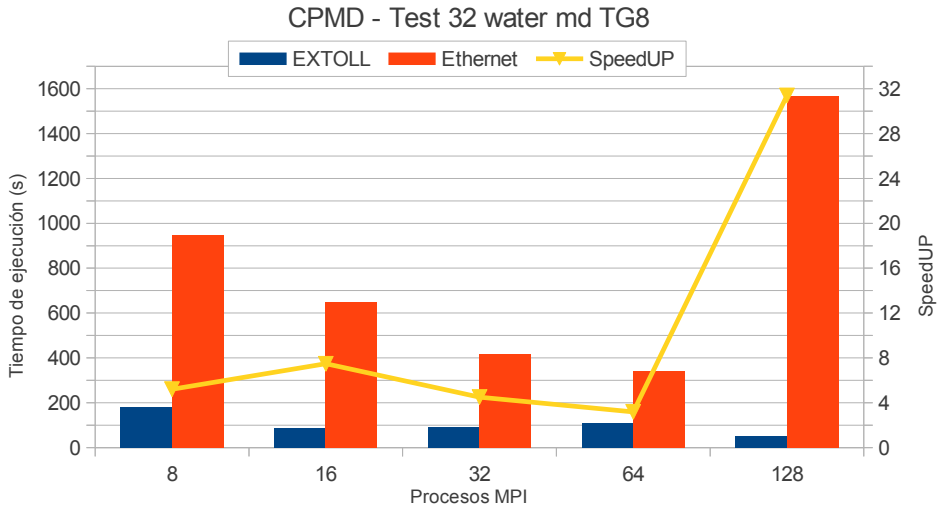

Figura 5.2: Simulación con 32 moleculas de agua sin optimización y 8 grupos de trabajo

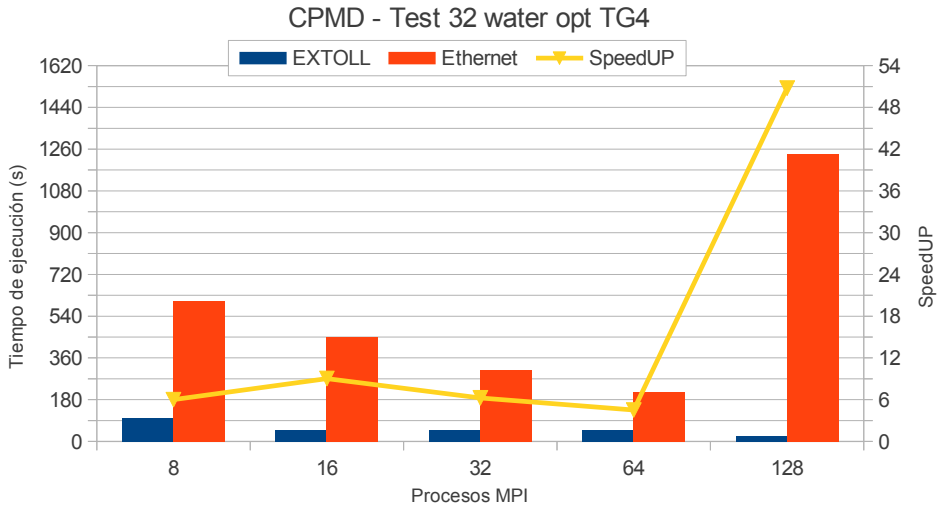

Figura 5.3: Simulación con 32 moleculas de agua con optimización y 4 grupos de trabajo

Tal como podemos apreciar en las Figuras 5.1, 5.2, 5.3 y 5.4 el tiempo de ejecución usando la red de interconexión EXTOLL es mucho menor que el obtenido al usar Ethernet. Además, para el caso de 128 procesos, Ethernet presenta un comportamiento extraordinariamente pobre. Este mal comportamiento de Ethernet en el caso de 128 procesos MPI se debe a que se están

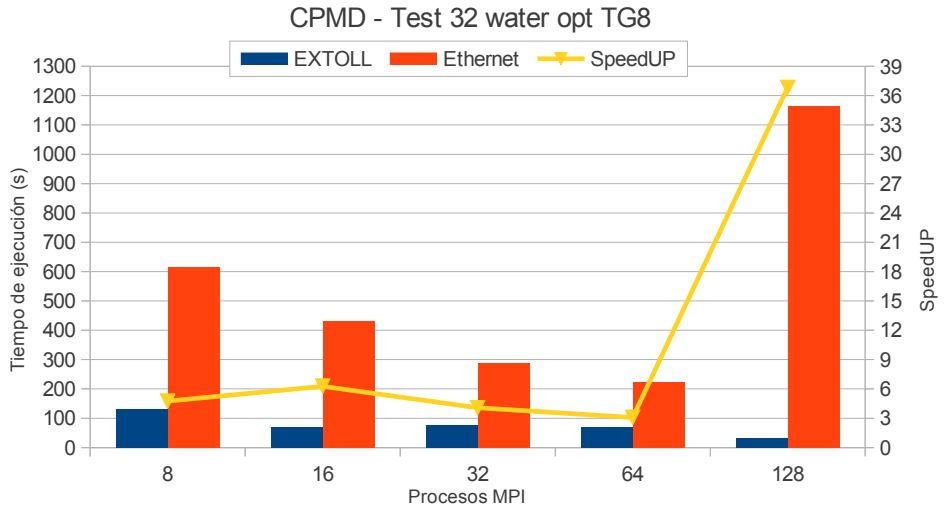

Figura 5.4: Simulación con 32 moleculas de agua con optimización y 8 grupos de trabajo

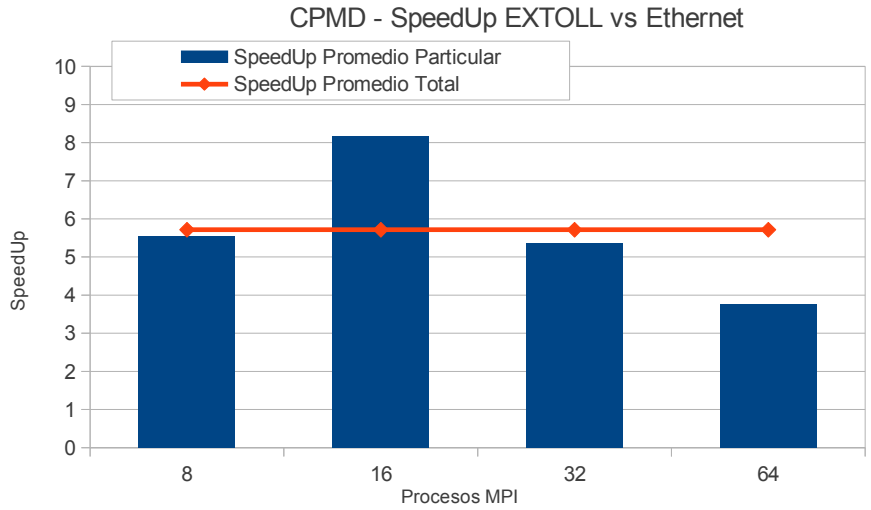

Figura 5.5: SpeedUp promedio parcial y total de EXTOLL sobre Ethernet para la aplicación CPMD

usando los 64 nodos del cluster, los dos switches por tanto, y existe un único cable que interconecta ambos switches. Por tanto este único cable representa un cuello de botella que estrangula y limita la velocidad de los intercambios de información entre ambos switches. Recordemos que la red Ethernet presente en el prototipo únicamente fué instalada con el propósito de servir a las

tareas de gestión y administración del mismo, nunca fué pensada para apoyo a la computación paralela. Por tanto este último caso, de 128 procesos no se tendrá en cuenta a la hora de realizar la comparación entre ambos sistemas de interconexión. Si observamos el eje Y secundario de estas figuras también podemos apreciar el SpeedUp de EXTOLL frente a Ethernet. A modo resumen podemos observar el SpeedUp promedio para todas las pruebas en la Figura 5.5 y el SpeedUp promedio general que nos muestra que las pruebas realizadas usando EXTOLL se ejecutan unas 5.8 veces más rápido que usando Ethernet.

5.3. DL_POLY

DL_POLY [9] y [21] es un paquete de subrutinas, programas y ficheros de datos, diseñado para facilitar las simulaciones de dinámica molecular de macro-moléculas, polímeros, sistemas iónicos y soluciones. Ha sido desarrollado en el Daresbury Laboratory por W. Smith y I.T. Todorov.

El propósito de DL_POLY es proporcionar software para la investigación académica barato, accesible y libre de consideraciones comerciales. Además los usuarios tienen acceso directo al código fuente para inspeccionarlo o modificarlo.

5.3.1. Evaluación DL_POLY

Para evaluar el comportamiento de la aplicación DL_POLY hemos utilizado un conjunto de pruebas desarrolladas para la versión 4 de dicha aplicación, en particular hemos utilizado las pruebas:

- Prueba 30 - Fe with Finnis-Sincair (metal) Potentials. Sistema de 250000 atómos de Hierro, simulado a 300 grados Kelvin usando Constant Number (N), Pressure (P), and temperature (T); T is regulated (NPT) berendsen y fuerzas Finnis-Sinclair sin electrostática.

- Prueba 40 - Ionic liquid dimethylimidazolium chloride. Sistema de 354816 iones, simulado a 400 grados Kelvin usando el conjunto de NPT Berendsen, a través de la dinámica tanto de partículas como de cuerpos rígidos con electroestática Solid-Phase MicroExtraction (SPME).

- Prueba 44 - Iron/Carbon alloy with EEmbedded Atom Method (EAM). Sistema de 294424 partículas, en el que se modela una aleación de acero formada por hierro y carbono en relación 35132-1651 utilizando un potencial EEAM. Simulando a 1000 K y 0 atmósferas mantenido en un conjunto Berendsen NPT.

- Prueba 46 - Iron/Cromium alloy with 2BEAM. Sistema de 256000 partículas, en el que se modela una aleación de acero formada por hierro y cromo en relación 27635-4365 utilizando un potencial 2BEAM de fuerza de campo. Simulando a 300 K y 0 atmósferas mantenido en un conjunto isocinético Evans Constant Number (N), Volume (V), and temperature (T); T is regulated (NVT).

Como ya hemos dicho en apartados anteriores cada una de estas pruebas ha sido ejecutada con un total de 16 procesos MPI por nodo, alcanzando un total de 1024 procesos MPI al usar los 64 nodos del prototipo. Finalmente decir que las pruebas ejecutadas usando Ethernet no han podido ser realizadas con 1024 procesos debido a un error reportado por OpenMPI.

Al igual que en la evaluación de CPMD, las Figuras 5.6, 5.7, 5.8 y 5.9 muestran los tiempos de ejecución usando EXTOLL y Ethernet. Además también muestran el SpeedUp alcanzado al usar EXTOLL respecto a la ejecución mediante Ethernet. Tal y como se observa en estas figuras el tiempo de ejecución al utilizar EXTOLL es ligeramente inferior al de Ethernet en las ejecuciones de las pruebas 30, 44 y 46. En la prueba 40 el tiempo de ejecución también se reduce al utilizar EXTOLL pero en este caso la reducción es mucho mayor que en los otros casos. Otro dato que se puede extraer del análisis de estos gráficos es que obtenemos un mejor nivel de escalabilidad al usar EXTOLL, ya que la curva del SpeedUp alcanzado entre EXTOLL y Ethernet es ascendente con el número de procesos.

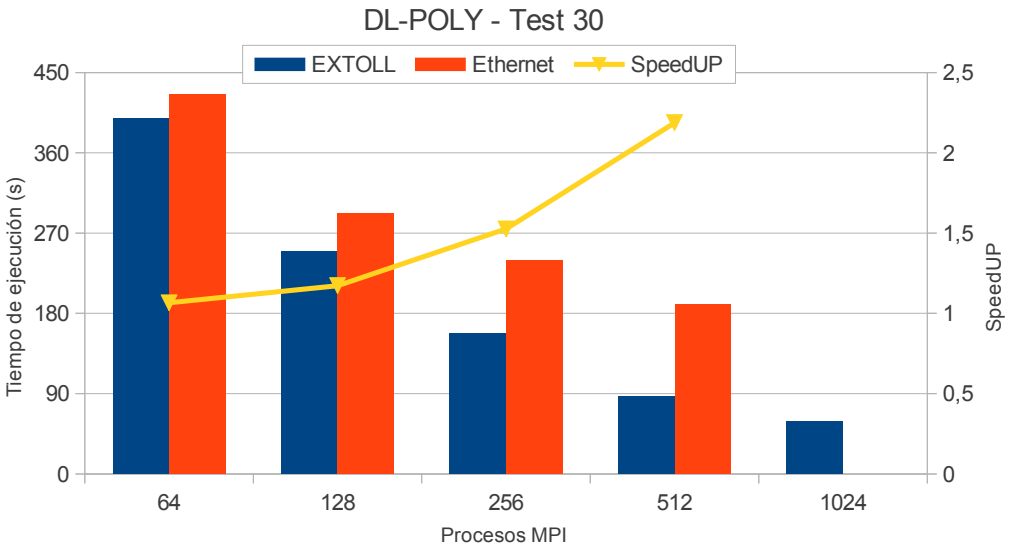

Figura 5.6: DL-POLY - Fe with Finnis-Sincair (metal) Potentials

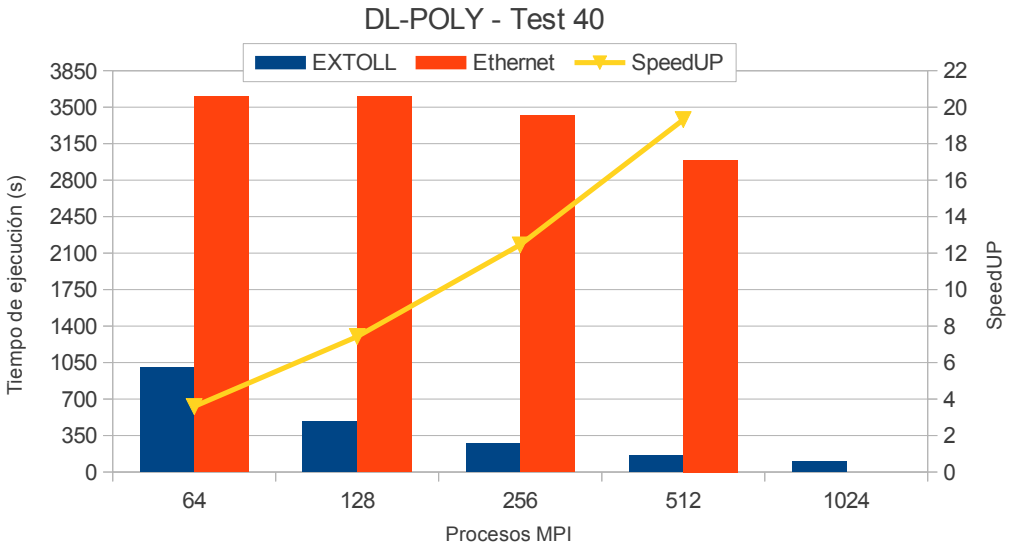

Figura 5.7: DL-POLY - Ionic liquid dimethylimidazolium chloride

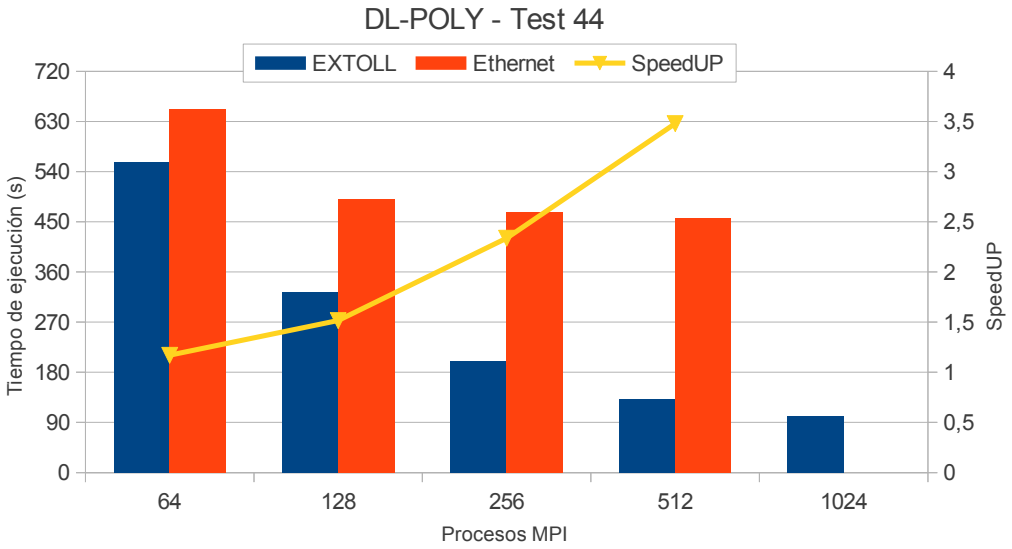

Figura 5.8: DL-POLY - Iron/Carbon alloy with EEAM

Finalmente en la Figura 5.10 podemos ver los valores de SpeedUp promedio parciales y el SpeedUp promedio total de EXTOLL sobre Ethernet para esta aplicación. Podemos ver que EXTOLL proporciona un SpeedUp igual a 4 respecto a la ejecución usando Ethernet.

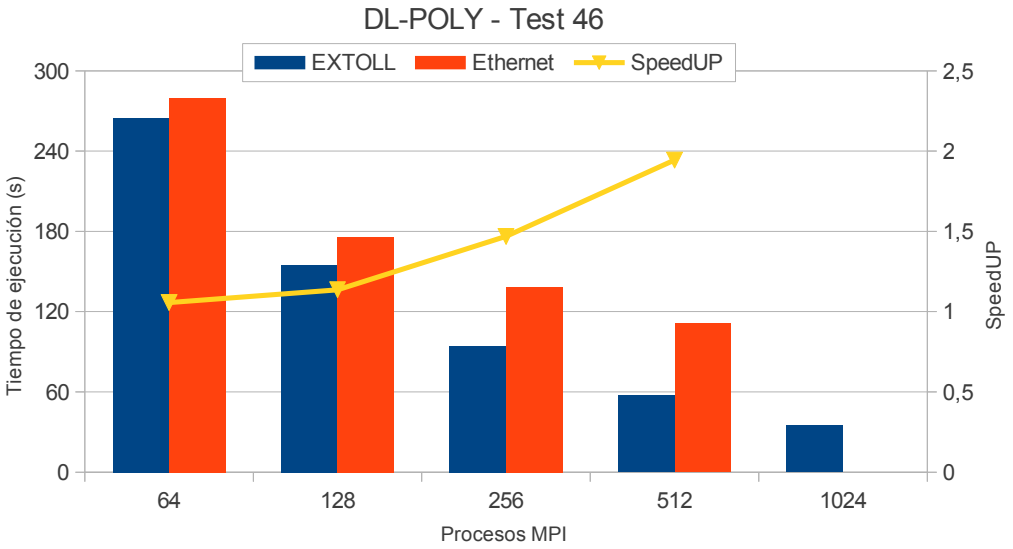

Figura 5.9: DL-POLY - Iron/Cromium alloy with 2BEAM

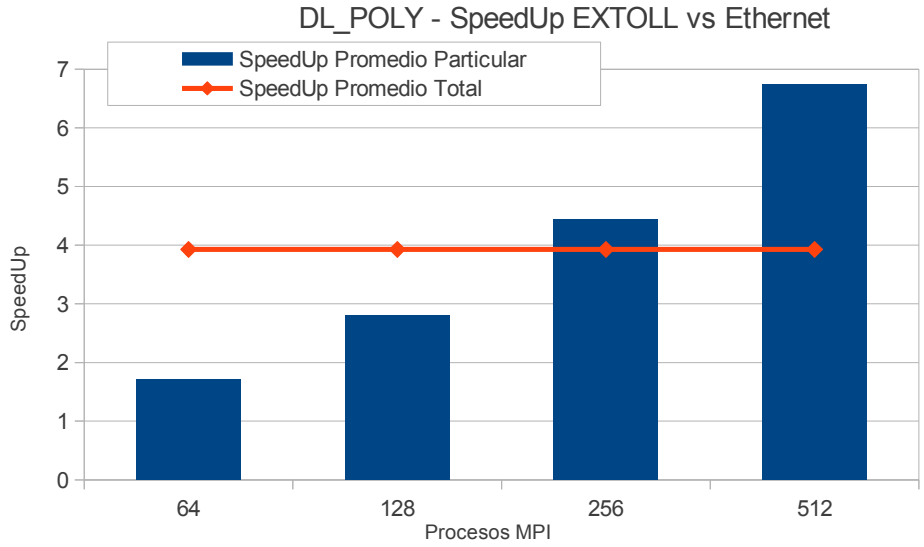

Figura 5.10: SpeedUp promedio particular y total de EXTOLL sobre Ethernet para la aplicación DL_POLY

5.4. LAMMPS

LAMMPS [8] es un código de dinámica molecular clásica que permite modelar un conjunto de partículas en un estado líquido, sólido o gaseoso.

Permite realizar un modelado atómico, polimérico, biológico, metálico, granular, o de sistemas de granularidad gruesa usando una gran variedad de campos de fuerza y de condiciones de contorno. En el sentido más general, LAMMPS integra las ecuaciones de Newton del movimiento a las colecciones de átomos, moléculas o partículas macroscópicas que interactúan a través de fuerzas de corto o largo alcance con una variedad de condiciones iniciales y / o de frontera. Después utiliza técnicas de descomposición espacial para dividir el dominio de simulación en pequeños sub-dominios 3D, cada uno de los cuales se asigna a un procesador.

Se puede ejecutar eficientemente en máquinas de un solo procesador pero está diseñado para ejecutarse en computadoras paralelas. Por tanto se ejecutará en cualquier máquina paralela capaz de compilar C++ y que tenga soporte para la librería de paso de mensajes MPI. Esto incluye máquinas paralelas distribuidas o de memoria compartida y agrupaciones de tipo Beowulf.

LAMMPS fue desarrollado originalmente por el Departamento de Energía de EE.UU. Cooperative Research and Development Agreement (CRADA) y 3 empresas. Actualmente es un código abierto disponible gratuitamente, distribuido bajo los términos de la Licencia Pública GNU y es distribuido por Laboratorios Nacionales Sandia de EE.UU.

5.4.1. Evaluación LAMMPS

La evaluación de esta aplicación ha sido realizada con 16 procesos MPI por nodo, alcanzando un total de 1024 procesos al ejecutarla con 64 nodos. Las pruebas ejecutadas han sido:

- EAM metallic solid benchmark. Cobre metálico sólido con potencial EAM. Simulación de 4 millones de átomos, 100 pasos, fuerza de corte de 4.95 Angstroms, 45 vecinos por átomo y NVE.
- Lennard-Jones liquid benchmark. Simulación de fluido atómico, con 4 millones de átomos, 100 pasos, fuerza de corte de 2.5 sigma, 55 vecinos por átomo y NVE.

Las Figuras 5.11 y 5.12 nos muestran los resultados obtenidos en las pruebas con esta aplicación. Al igual que en el caso de CPMD debemos descartar el test de 1024 procesos con Ethernet debido al cuello de botella formado en la comunicación entre los dos switches. Una vez descartado este caso, observamos un comportamiento muy similar al observado en la aplicación DL_POLY, es decir, tenemos un tiempo de ejecución y un nivel de escalabilidad mucho mejores al realizar las pruebas con EXTOLL que al realizarlas

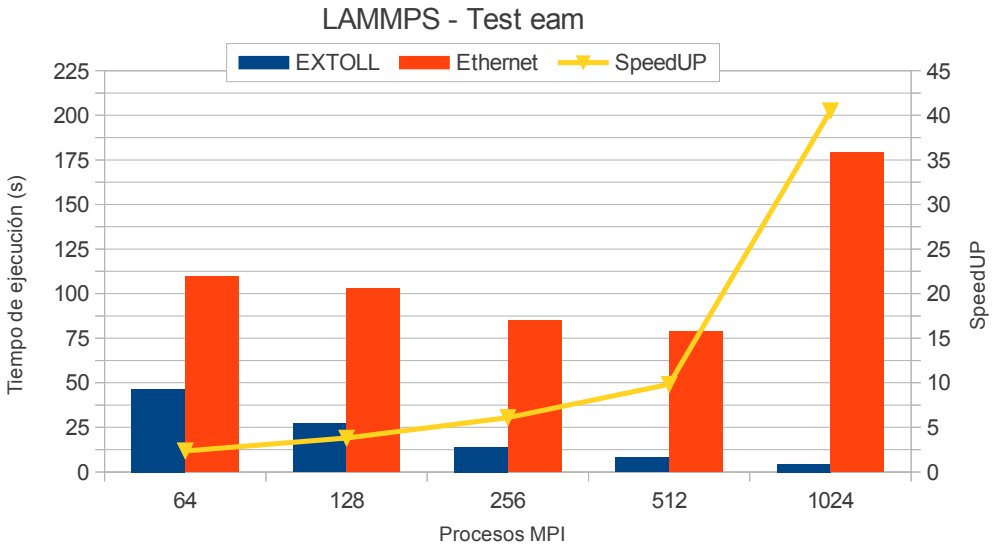

Figura 5.11: LAMMPS - Cobre metálico sólido con potencial EAM

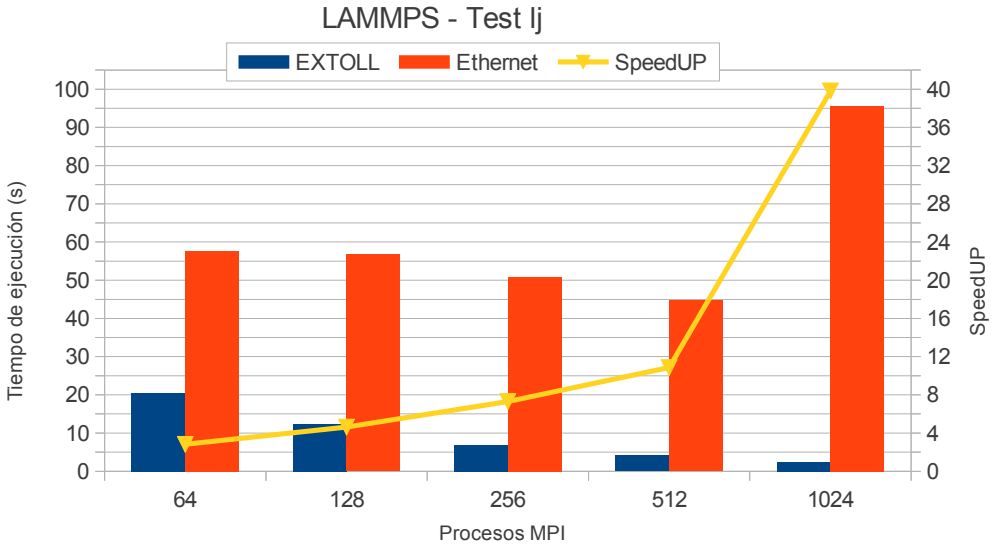

Figura 5.12: LAMMPS - Simulación de fluído atómico Lennard-Jones

con Ethernet. En la Figura 5.13 podemos observar los niveles de SpeedUp promedio particulares así como el total de EXTOLL frente a Ethernet. Se puede apreciar que en promedio las pruebas realizadas mediante EXTOLL se ejecutan 6 veces más rápidamente que usando Ethernet.

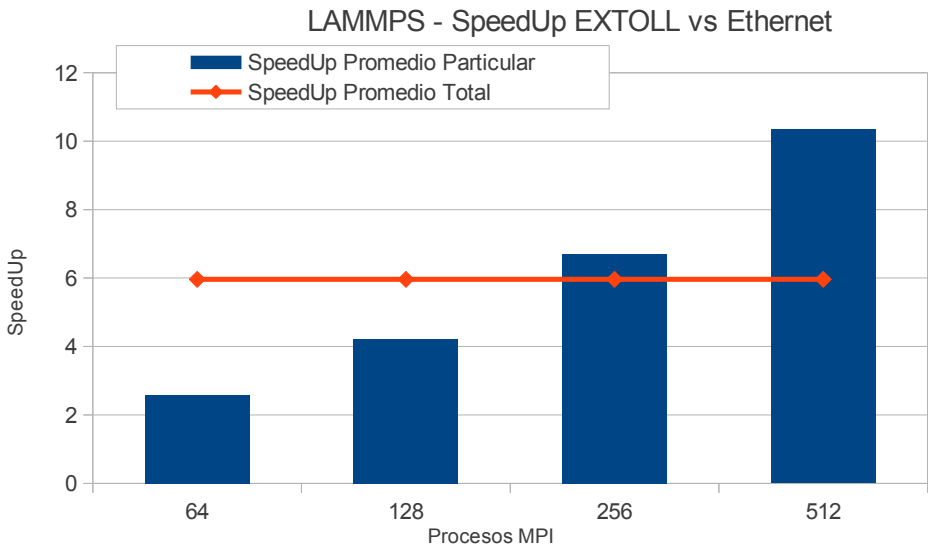

Figura 5.13: SpeedUp promedio particular y total de EXTOLL sobre Ethernet para la aplicación LAMMPS

5.5. GROMACS

GROMACS [2] es un paquete versátil para realizar simulaciones de dinámica molecular, es decir, simular las ecuaciones de Newton del movimiento para sistemas de cientos a millones de partículas. Está diseñado principalmente para moléculas bioquímicas como proteínas, lípidos y ácidos nucleicos que tienen un montón de complicadas interacciones, pero debido a que GROMACS es extremadamente rápido en el cálculo de las interacciones no enlazantes (que por lo general dominan las simulaciones) muchos grupos también lo están utilizando para la investigación sobre sistemas no biológicos, por ejemplo, polímeros.

GROMACS fue desarrollado por primera vez en el grupo de Herman Berendsen, departamento de Química Biofísica de la Universidad de Groningen. Pero actualmente representa un esfuerzo de equipo, con contribuciones de varios desarrolladores actuales y antiguos de todo el mundo.

GROMACS es software libre, y está disponible bajo la licencia GNU Lesser General Public License.

5.5.1. Evaluación GROMACS

Esta aplicación ha sido ejecutada con un solo proceso MPI por nodo, alcanzando un total de 64 procesos al ejecutarla en todos los nodos del cluster.

Únicamente ha sido evaluada con 64 procesos MPI porque las pruebas utilizadas durante esta evaluación no permitían una mayor escalabilidad. Todos los tests realizados están basados en la proteína de dihidrofolato reductasa 159-residuo con aproximadamente 7000 particulas de agua como disolvente. Los test utilizados han sido:

- Simulación con fuerzas electroestáticas PME y 1000 pasos.
- Simulación de 1000 pasos con un campo de reacción de 1 nm.
- Simulación de 1000 pasos con un campo de reacción de 2 nm.

Las Figuras 5.14, 5.15 y 5.16 vuelven a mostrar un comportamiento muy pobre para el caso de 64 nodos y Ethernet. Sin embargo para esta aplicación el rendimiento de EXTOLL es muy parecido al ofrecido por Ethernet. Esto puede ser debido a un menor peso de la comunicación respecto a las aplicaciones vistas anteriormente. Si analizamos la Figura 5.17 podemos ver como tanto los niveles de SpeedUp particulares como el total están muy cerca de 1 indicando un rendimiento muy parecido usando cualquiera de los dos sistemas de interconexión.

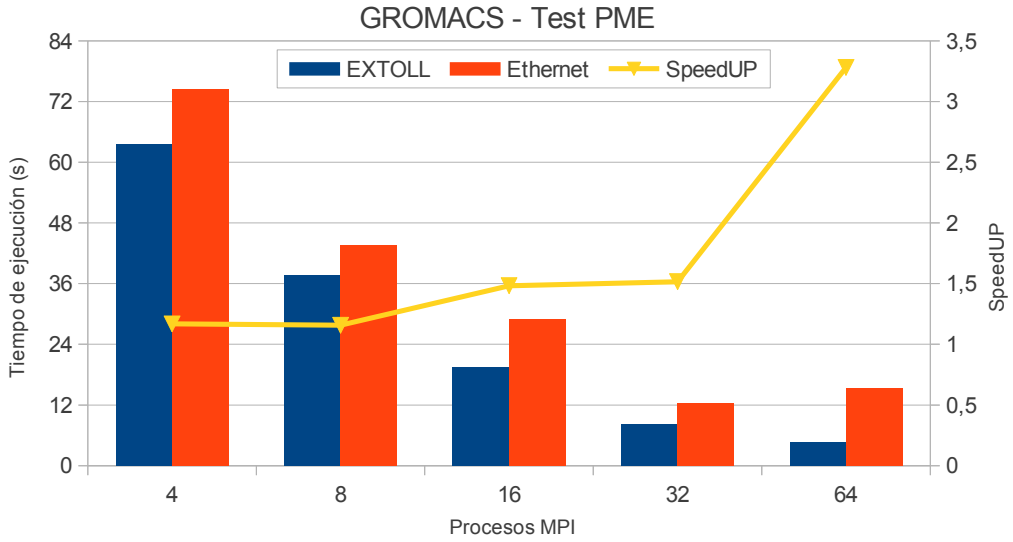

Figura 5.14: GROMACS - Prueba PME

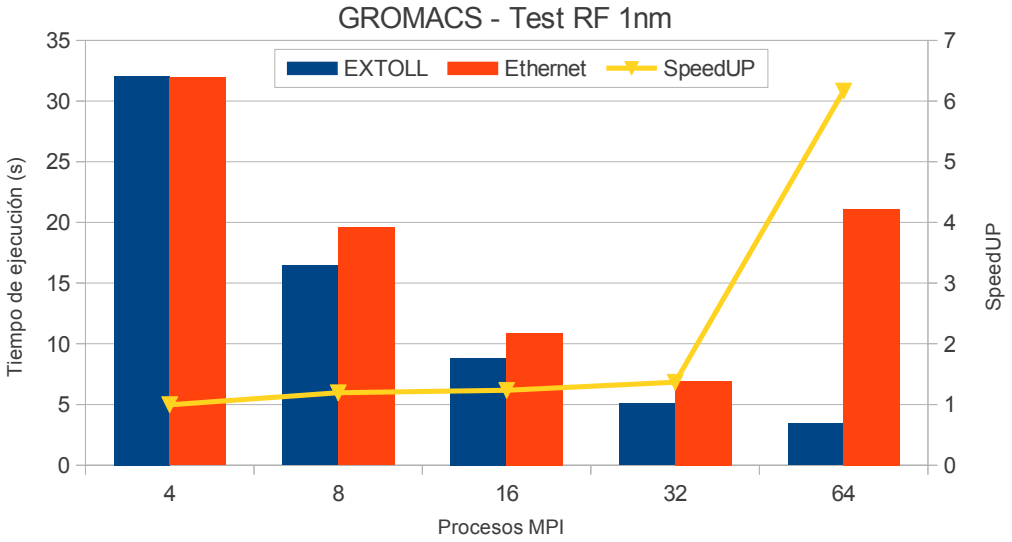

Figura 5.15: GROMACS - Prueba Reaction-Field 1nm

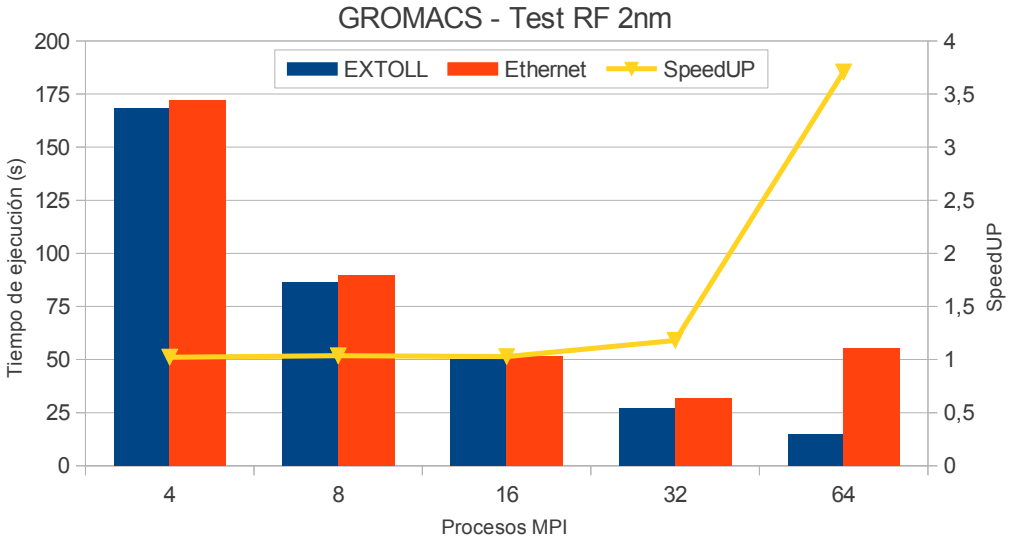

Figura 5.16: GROMACS - Prueba Reaction-Field 2nm

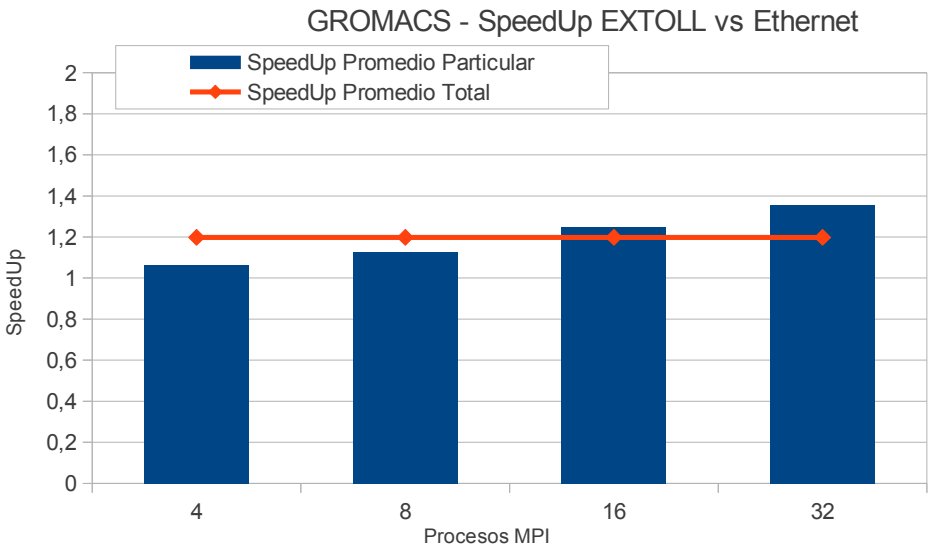

Figura 5.17: SpeedUp promedio particular y total de EXTOLL sobre Ethernet para la aplicación GROMACS

5.6. Comparación conjunta

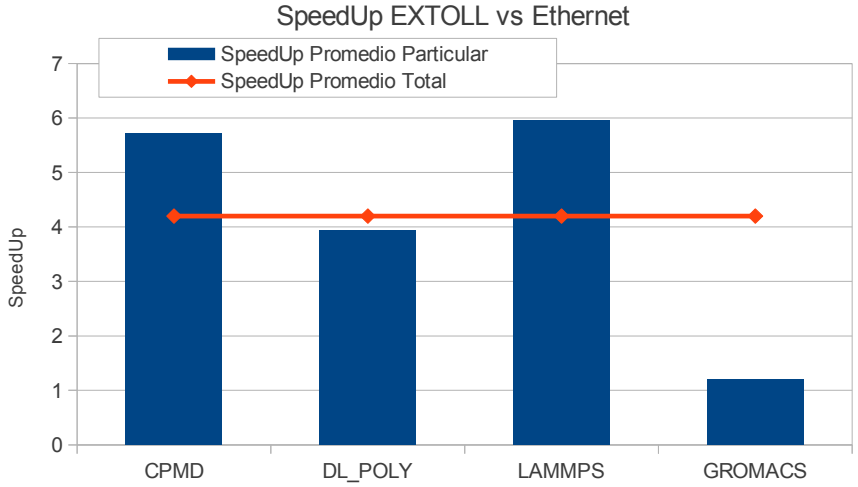

Figura 5.18: SpeedUp promedio particular y total de EXTOLL sobre Ethernet analizando las cuatro aplicaciones

Finalmente realizaremos una comparación del rendimiento conjunta, utilizando los datos obtenidos al ejecutar cada una de las aplicaciones. Para

ello simplemente calcularemos el SpeedUp alcanzado por EXTOLL sobre Ethernet al ejecutar los diferentes tests de las cuatro aplicaciones, es decir, calcularemos el SpeedUp promedio global a partir los SpeedUp's de cada una de las aplicaciones. La Figura 5.18 muestra exactamento esto, por lo tanto podemos afirmar que la ejecución de este conjunto de aplicaciones usando EXTOLL es acelerada alrededor de 4 veces respecto a la ejecución usando Ethernet.

Capítulo 6

Comparación de prestaciones en tiempo real

En este capítulo se muestra un método para apreciar visualmente la diferencia de rendimiento entre el uso de EXTOLL y Ethernet en tiempo real. El método utilizado consiste en la adaptación de una rutina paralela, que genera una porción del conjunto M o conjunto de Mandelbrot, para que al ejecutarla, usando EXTOLL o Ethernet, se aprecien las diferencias debidas al distinto nivel de rendimiento de ambos sistemas de interconexión.

El conjunto de Mandelbrot es uno de los conjuntos fractales más estudiados. En líneas generales se compone de un conjunto de puntos para los cuales cierta operación matemática da siempre resultados menores que un cierto valor, están acotados, y los puntos que no forman parte de él tienden al infinito al aplicarles la misma operación matemática. Entonces, si representamos los puntos pertenecientes al conjunto en negro y los no pertenecientes con un color que dependa de la velocidad con la que éstos tienden a infinito al aplicarles la operación matemática, obtenemos imágenes espectaculares.

6.1. Introducción

En el capítulo anterior se ha realizado un amplio estudio de las prestaciones obtenidas al ejecutar, usando las redes EXTOLL y Ethernet, una serie de aplicaciones reales. Este estudio nos ha permitido comparar el rendimiento de ambos sistemas de comunicación y cuantificar la mejora obtenida al usar EXTOLL respecto a Ethernet.

El objetivo de este capítulo es mostrar esta diferencia de rendimiento en tiempo real, para ello se generará una porción del conjunto M o conjunto de Mandelbrot usando EXTOLL y Ethernet al mismo tiempo. De esta forma se apreciará la diferencia de prestaciones a través de la diferente velocidad de generación de las imágenes.

6.2. Conjunto de Mandelbrot

El conjunto de Mandelbrot es uno de los conjuntos fractales más estudiados y debe su nombre al matemático polaco Benoît Mandelbrot. Tal como puede verse en [4] este conjunto es, en líneas generales, el conjunto de puntos para los cuales cierta operación matemática da siempre resultados menores que un cierto valor. Más concretamente: un número complejo z_0 (un punto del plano) está en el conjunto de Mandelbrot si la sucesión de puntos siguiente:

$$z_0$$

$$z_1 = z_0^2 + z_0$$

$$z_2 = z_1^2 + z_0$$

$$\dots$$

$$z_{n+1} = z_n^2 + z_0$$

está acotada. Si esta sucesión de puntos no está acotada, o lo que es lo mismo, sus valores crecen y crecen indefinidamente, el punto no está en el conjunto.

Los puntos del conjunto de Mandelbrot son los que aparecen en la Figura 6.1. Los que no pertenecen al conjunto no tienen por qué representarse, aunque lo que le da ese tremendo juego a este conjunto es representar con colores la velocidad con la que estos puntos que no pertenecen al conjunto se acercan a infinito, como aparece en la Figura 6.2. Los puntos para los que su sucesión crece muy rápido están representados en color rojo intenso. El rojo va tornándose más suave conforme la velocidad de crecimiento es menor. Los puntos muy cercanos al conjunto (en blanco) son puntos para los que ha hecho falta calcular muchísimos valores de la sucesión asociada para ver

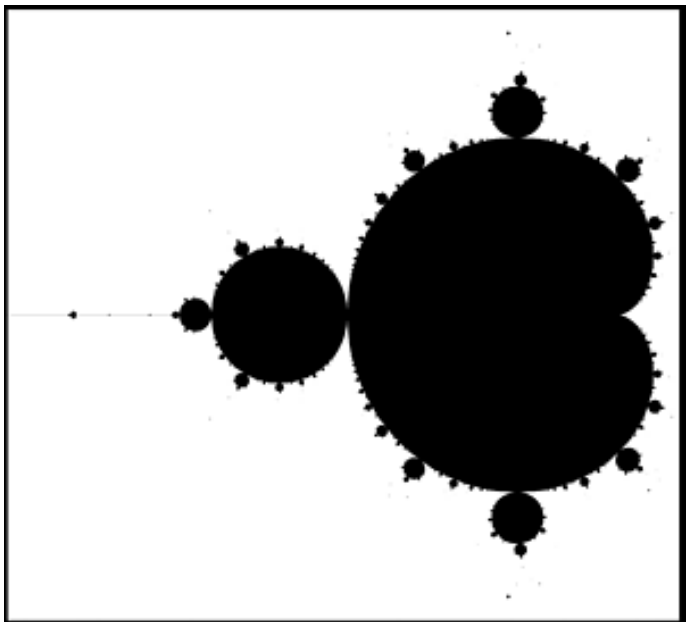

Figura 6.1: Puntos pertenecientes al conjunto de Mandelbrot

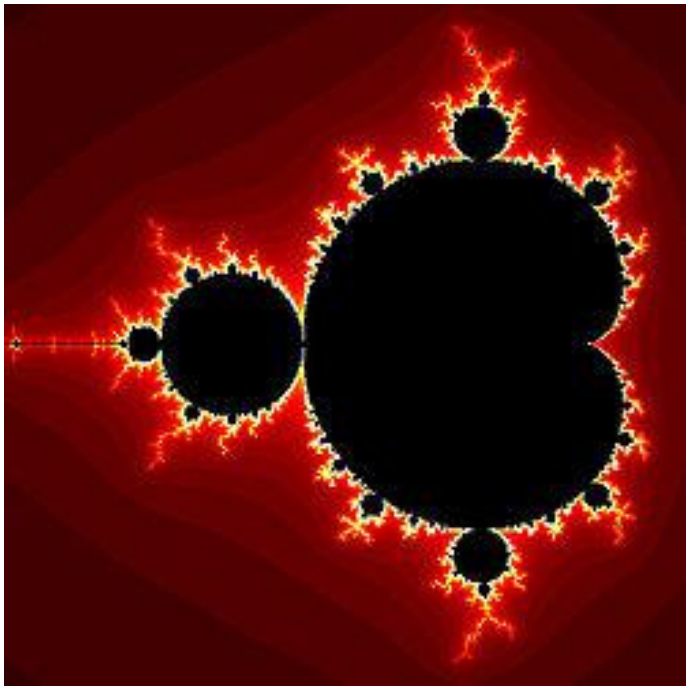

Figura 6.2: Conjunto de Mandelbrot con colores

que no está acotada. Este juego de colores provoca que al hacer zoom en el conjunto, las imágenes que se crean sean de una belleza inusitada. Además, este zoom hace que nos demos cuenta de que éste es un conjunto fractal, ya que al acercarnos vemos que el propio conjunto contiene copias exactas de sí mismo.

6.3. Algoritmo

El código paralelo MPI usado para generar el conjunto de Mandelbrot forma parte del trabajo mostrado en [22] y [13]. Este programa calcula y muestra la totalidad o parte del conjunto de Mandelbrot. Por defecto, se examinan todos los puntos en el plano complejo que tienen tanto la parte real como imaginaria comprendidas entre -2 y 2. Los parámetros de línea de comandos permiten realizar zoom en una parte específica de este rango, así como especificar un número determinado de iteraciones.

6.3.1. Funcionamiento del algoritmo

En el código original se dividen las líneas de la imagen a generar entre el número de procesos MPI disponibles, luego cada proceso MPI calcula los valores de color para los píxeles de las líneas que le han sido asignadas. Una vez que estos valores han sido calculados son enviados al proceso maestro que se encarga de dibujarlos en pantalla.

6.3.2. Modificación realizada

Hemos realizado una pequeña modificación en el código original con el objetivo de incrementar el peso de la comunicación en relación con el del cómputo, ya que queremos comparar de forma visual el rendimiento de dos redes de interconexión. Esta modificación consiste en repartir el número de píxeles de la imagen entre el número de procesos MPI existentes y cada vez que un proceso MPI calcula el valor de uno de los píxeles que tiene asignados se lo comunica al proceso maestro para que lo dibuje en pantalla.

6.3.3. Resultados

La ejecución de esta aplicación paralela permitirá poder comparar en tiempo real, y de una forma muy visual, el rendimiento de diferentes sistemas de interconexión, en nuestro caso EXTOLL y Ethernet. La Figura 6.3

muestra dos ejecuciones simultáneas de 128 procesos MPI cada una, la superior mediante EXTOLL y la inferior mediante Ethernet, de la aplicación en nuestro prototipo.

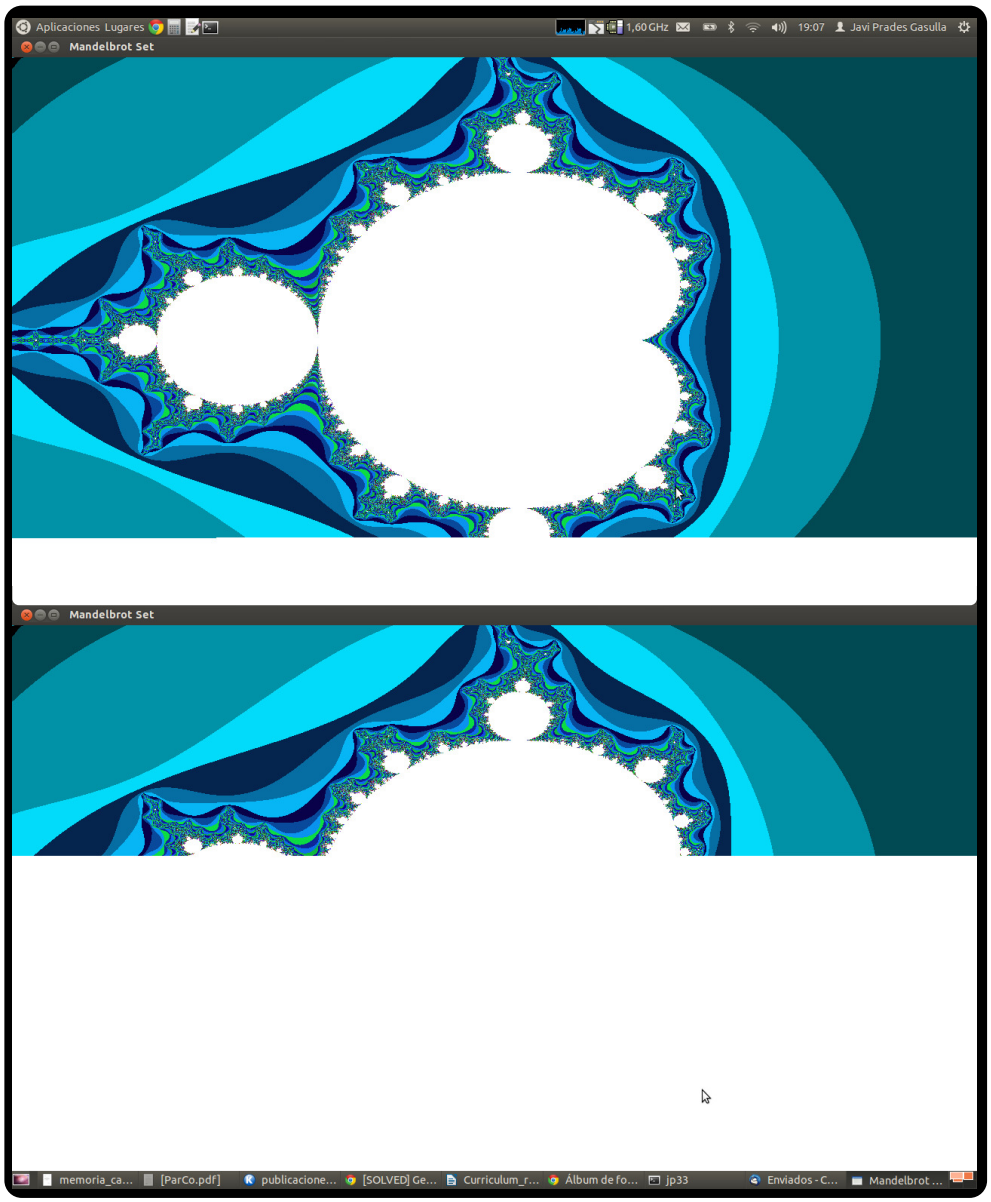

Figura 6.3: Generación del fractal de Mandelbrot

Capítulo 7

Control de Flujo

Durante la realización de las pruebas de comparación del rendimiento de EXTOLL y Ethernet, mediante el uso de aplicaciones reales, empezamos a observar que algunos de los nodos del cluster se bloqueaban de forma inesperada, al ejecutar pruebas de gran tamaño con EXTOLL. Después de una ardua investigación, concluimos que la causa de los bloqueos se debía a un mal uso del mecanismo de control de flujo a nivel de HyperTransport. Por este motivo se desarrollaron dos mecanismos, uno estático y otro dinámico, de control de flujo de extremo a extremo y basados en créditos para EXTOLL. El desarrollo de estos mecanismos queda fuera del alcance de este proyecto, pero debido a que estos mecanismos han sido utilizados en todos los experimentos realizados usando EXTOLL, hemos decidido explicar en un breve capítulo las principales características de cada uno.

El mecanismo estático es muy eficiente si el patrón de comunicación está muy equilibrado, es decir, cuando todos los procesos se comunican con todos los restantes. Sin embargo este mecanismo pierde mucha eficiencia cuando este patrón deja de ser equilibrado. Debido a esto se evolucionó el diseño estático a otro dinámico, capaz de adaptarse al patrón de comunicación subyacente en tiempo de ejecución, que presenta una eficiencia similar a la del estático cuando el patrón es equilibrado y mucho mejor cuando deja de serlo.

Después de analizar y comparar la eficiencia de ambos mecanismos de control de flujo, podemos establecer que el mecanismo dinámico es 4 veces más eficiente al ejecutar los benchmarks de Intel para MPI.

7.1. Necesidad de un mecanismo de control de flujo

Durante la realización de las pruebas de comparación del rendimiento de EXTOLL y Ethernet, mediante el uso de aplicaciones reales, empezamos a observar bloqueos inesperados en los nodos, al ejecutar pruebas grandes con EXTOLL. En un primer momento atribuimos estos bloqueos a un mal comportamiento de los transceivers y/o los enlaces de comunicación de la red EXTOLL. Sin embargo, tras una ardua labor de investigación descartamos esta posible fuente del error. Después de comunicar el problema a los diseñadores de EXTOLL de la universidad de Heidelberg y tras una investigación conjunta, descubrimos que el origen de dichos bloqueos podía ser debido a que el mecanismo de control de flujo existente, a nivel de HyperTransport, no gestionaba correctamente los envíos realizados a través de VELO. Por tanto se decidió diseñar un nuevo mecanismo de control de flujo software de extremo a extremo basado en créditos que permitiera resolver este problema. El diseño de este mecanismo de control de flujo es muy complejo y su explicación queda fuera del alcance de lo que pretende mostrar este proyecto, pero creemos que es importante mostrar en este trabajo algunos de los aspectos más importantes de este nuevo mecanismo de control de flujo ya que éste ha sido usado en todas las pruebas presentes en este documento.

7.2. Sobre los mecanismos de control de flujo

Básicamente, un mecanismo de control de flujo impide que los receptores lentos, o los receptores que están ocupados haciendo otras tareas, vean sus buffers de recepción desbordados debido a transmisores que transmiten demasiado rápido. Téngase en cuenta que si los buffers de recepción tuviesen un tamaño ilimitado, entonces no existiría esta preocupación por el desbordamiento y, por lo tanto, no sería necesario un mecanismo de control de flujo. Si bien esto, obviamente, no es factible ya que el uso de grandes buffers significaría dedicar una gran cantidad de recursos de memoria a algo que no pertenece a la aplicación en ejecución sino a la infraestructura de comunicación subyacente. Por lo tanto, el costo de estos recursos debe mantenerse tan bajo como sea posible. Además, nótese que estos recursos de memoria dependen del número de procesos implicados. En consecuencia, los mecanismos de control de flujo pueden ser vistos como técnicas que establecen un límite máximo a los recursos de memoria utilizados por la capa de comunicación. Sin embargo, esta limitación en los recursos de almacenamiento puede repercutir en un aumento del tiempo de ejecución por dos razones: primero,

algunos procesos pueden estancarse debido a la falta de espacio en el lado del receptor. En segundo lugar, el protocolo de control de flujo en sí mismo introduce una sobrecarga computacional, así como un aumento de tráfico adicional en la red, debido a la necesidad de comunicar el estado de los buffers de recepción. Por lo tanto, la eficiencia de un mecanismo de control de flujo puede ser vista como un compromiso entre los recursos que necesita y la sobrecarga que genera.

Con el objetivo de realizar un control de flujo eficiente, partiremos de un mecanismo de control de flujo capaz de gestionar los recursos de buffering de forma estática, que posteriormente evolucionaremos a otro más eficiente que los gestionará en tiempo de ejecución. Ambos protocolos se basan en la utilización de créditos para controlar la cantidad de recursos disponibles en los buffers de recepción, aunque la versión dinámica gestiona los créditos de una manera más flexible, logrando así un mejor rendimiento. Es importante destacar que ambos mecanismos, estático y dinámico, han sido implementados totalmente dentro de la librería libvelo, siendo transparentes, por lo tanto, a las capas de software superiores, como MPI o GASNET. Estas capas no requerirán ninguna modificación para beneficiarse de la implementación de estos nuevos protocolos de control de flujo de extremo a extremo.

7.3. Control de flujo estático

En el mecanismo de control de flujo estático el espacio de los buffers de recepción de cada uno de los procesos receptores es repartido de forma equitativa entre todos sus posibles emisores. Por tanto este mecanismo de control de flujo será muy eficiente si la comunicación entre los distintos procesos es muy equilibrada, es decir, todos los procesos se comunican con todos los restantes. A medida que el patrón de comunicación va siendo más localizado, un proceso se comunica con un reducido conjunto de los procesos restantes. La eficiencia de este mecanismo estático se va viendo comprometida, ya que muchos de los recursos de buffering no son usados, sino que los tienen asignados emisores que realmente no van a transmitir en ningún momento.

En la Figura 7.1 se muestra la sobrecarga generada, respecto al tiempo de ejecución de referencia [1], de cuatro patrones diferentes de tráfico, que van desde un patrón muy equilibrado hasta otro muy poco equilibrado, o lo que

[1]Este tiempo de ejecución de referencia ha sido calculado sin ningún mecanismo de control de flujo, pero utilizando una gran cantidad de recursos de buffering, los cuales nos garantizan la inexistencia de desbordamientos. Por lo tanto este tiempo de referencia puede considerarse como el mejor que puede obtenerse en nuestro sistema y con la tecnología de red EXTOLL.

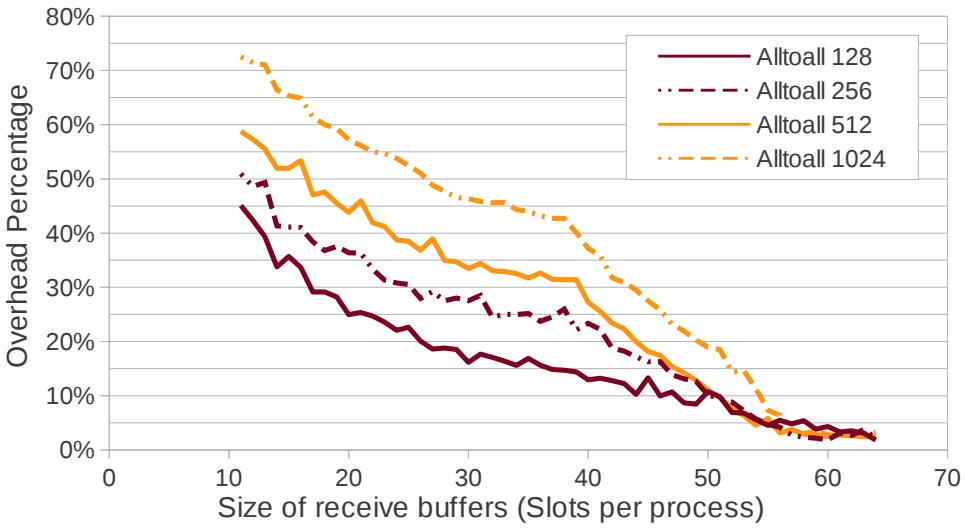

Figura 7.1: Sobrecarga generada por el control de flujo estático para cuatro patrones de tráfico diferentes

es lo mismo, un patrón muy localizado. Estos patrones son:

- Alltoall 1024 - Aplicación de 1024 procesos MPI en la que todos se ven involucrados en una operación alltoall (patrón de comunicación muy equilibrado)
- Alltoall 512 - Aplicación de 1024 procesos MPI en la que el 50 % de los procesos están involucrados en un alltoall mientras que el 50 % restante espera en un barrier.
- Alltoall 256 - Aplicación de 1024 procesos MPI en la que el 25 % de los procesos están involucrados en un alltoall mientras que el 75 % restante espera en un barrier.
- Alltoall 128 - Aplicación de 1024 procesos MPI en la que el 12,5 % de los procesos están involucrados en un alltoall mientras que el 87,5 % restante espera en un barrier (patrón de comunicación muy localizado)

Analizando la Figura 7.1 podemos ver que sea cual sea el patrón de tráfico, el mecanismo de control de flujo estático necesita la misma cantidad de recursos, alrededor de 60 slots por posible emisor, para que la sobrecarga sea despreciable (inferior al 5 %).

7.4. Control de flujo dinámico

Una vez analizados los problemas que presenta un reparto estático del espacio de los buffers de recepción se evolucionó el diseño a un nuevo mecanismo de control de flujo dinámico. Este mecanismo es capaz de gestionar los recursos de buffering de forma dinámica, en tiempo de ejecución, para así adaptarse a cualquier patrón de comunicación subyacente. La Figura 7.2

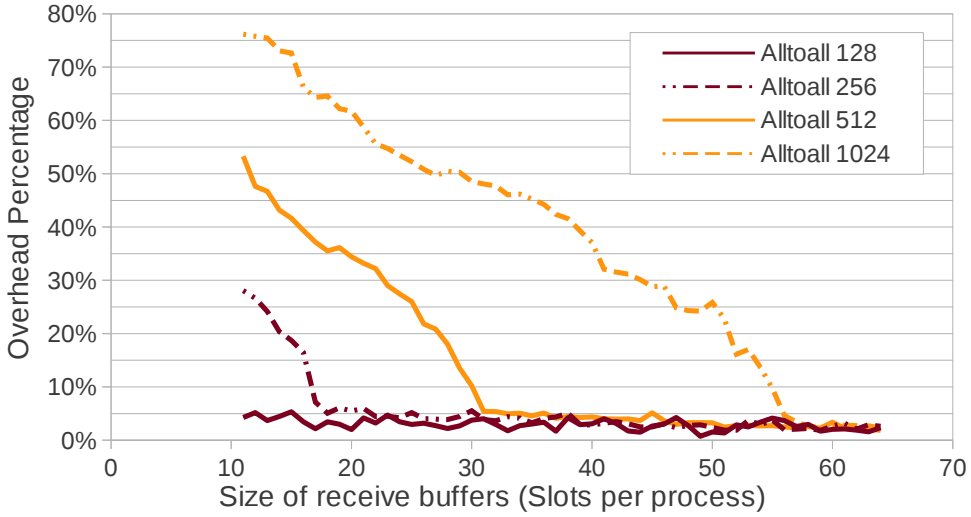

Figura 7.2: Sobrecarga generada por el control de flujo dinámico para cuatro patrones de tráfico diferentes

muestra exactamente la misma prueba que la mostrada en la Figura 7.1 pero en este caso el mecanismo de control de flujo utilizado ha sido el dinámico. Si comparamos los resultados de ambas figuras podemos apreciar que el control de flujo dinámico tiene un comportamiento muy similar al estático cuando el patrón de tráfico es muy equilibrado. Pero en cambio muestra unos resultados mucho mejores cuando el patrón de tráfico es más localizado.

7.5. Resultados

En [20] puede verse una exhaustiva comparación de la eficiencia de los mecanismos de control de flujo, estático y dinámico, presentados en esta sección. Esta comparación fué realizada usando el conjunto de benchmarks para MPI de INTegrated ELectronics (INTEL) citados en el Capítulo 4. La Figura 7.3 muestra la sobrecarga de los mecanismos dinámico y estático en el tiempo de ejecución de estos benchmarks en función de los recursos de

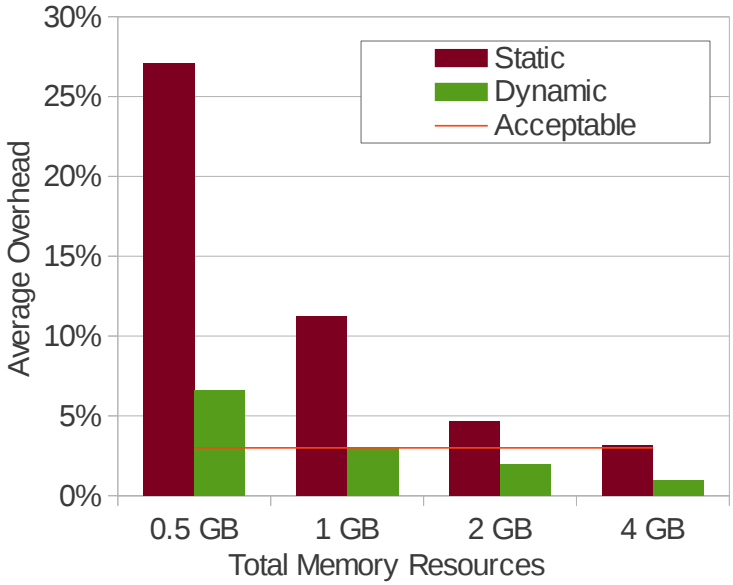

Figura 7.3: Sobrecarga media generada por los mecanismos de control de flujo, estático y dinámico, durante la ejecución de la suite de benchmarks de INTEL para MPI

memoria asignados. Tal como puede observarse, el mecanismo de control de flujo dinámico necesita 4 veces menos recursos de memoria que el dinámico para obtener una eficiencia similar. El mecanismo estático necesita un total de 4GB de memoria repartidos en todo el prototipo para ofrecer una sobrecarga aceptable, mientras que el dinámico con 1GB de memoria presenta la misma sobrecarga media. Podemos establecer, por tanto, que el mecanismo dinámico es 4 veces más eficiente que el estático al ejecutar este conjunto de pruebas.

Capítulo 8

Conclusiones

En la computación de altas prestaciones las redes de interconexión tienen un papel protagonista, ya que la mayoría de sistemas de supercomputación están basados en clusters formados por miles de nodos. Esto repercute en que las prestaciones del sistema dependan en gran medida del sistema de interconexión empleado.

En este trabajo se ha comparado el rendimiento de un nuevo sistema de interconexión, EXTOLL, con otro que es muy popular en los sistemas cluster, Gigabit Ethernet. Según nuestras pruebas, EXTOLL presenta en promedio una latencia cinco veces menor que Gigabit Ethernet y un ancho de banda real que ronda los 470MB/s, que contrastan con los 100MB/s que obtenemos con Gigabit Ethernet. La comparación de ambos sistemas mediante aplicaciones reales nos proporciona un SpeedUp promedio de EXTOLL sobre Gigabit Ethernet de 4x y una mejor escalabilidad en la mayoría de las pruebas.

Aunque a partir de esta comparación pueda parecer que EXTOLL es un sistema de interconexión muy apropiado, que lo es, esta comparación no es del todo justa, ya que Gigabit Ethernet comenzó a usarse por el año 2000, mientras que la versión de EXTOLL utilizada es del año 2009 y, aunque actualmente Gigabit Ethernet sigue estando en el Top500, cada año va perdiendo peso debido a que los nuevos sistemas están empezando a utilizar redes de interconexión más modernas, como InfiniBand [6]. Además, si se realizara una comparación precio/prestaciones entre EXTOLL y Gigabit Ethernet posiblemente la balanza se decantaría hacia el lado de Ethernet. Por tanto deberíamos comparar EXTOLL frente a los sistemas de interconexión más caros y modernos, como la última versión de InfiniBand Connect-IB con doble puerto, que proporciona un ancho de banda de 12.5GB/s y una latencia de 1µs (valores teóricos). Como vemos EXTOLL es muy inferior a esto. Sin embargo no debemos olvidar que la versión utilizada de EXTOLL está implementada en una FPGA, es del año 2009 y no deja de ser un prototipo.

Por lo que se espera que cuando se complete el diseño en el chip ASIC sus valores de rendimiento se asemejen a los de InfiniBand.

Como línea de investigación futura, cabría destacar la realización de una comparación similar a la realizada en este trabajo pero mediante el uso de la versión definitiva de EXTOLL y una de las más recientes de InfiniBand, así como una posible comparación entre la eficiencia de los mecanismos de control de flujo presentes tanto en EXTOLL como InfiniBand, ya que la eficiencia del mecanismo de control de flujo que hemos implementado para EXTOLL es muy alta.

Finalmente destacar que a raiz de la realización de todo este trabajo han surgido las siguientes publicaciones:

J. Prades, F. Silla, H. Fröning, and J. Duato. **A Network Insight of a 32-node EXTOLL Prototype.** In *ACACES 2010*, (ISBN 978-90-382-1631-7)

J. Prades, F. Silla, H. Fröning, and J. Duato. **Exploring a 512-core 32-node EXTOLL Network.** In *XXI Jornadas de Paralelismo 2010*, (ISBN 978-84-92812-49-3)

J. Prades, F. Silla, H. Fröning, M. Nüssle, and J. Duato. **Efficient Buffer Usage Through a New Flow-Control Mechanism.** In *XXIII Jornadas de Paralelismo 2012*, (ISBN 978-84-695-4471-6)

J. Prades, F. Silla, H. Fröning, M. Nüssle, and J. Duato. **A New End-to-End Flow-Control Mechanism for High Performance Computing Clusters.** In *CLUSTER 2012*, (ISBN 978-0-7685-4807-4)

J. Prades, F. Silla, H. Fröning, M. Nüssle, and J. Duato. **On the Design of a New Dynamic Credit-Based End-to-End Flow-Control Mechanism for HPC Clusters.** *Submitted to Parallel Computing Journal 2014*

Apéndices

Apéndice A

Configuración e instalación del software de EXTOLL

En este apéndice se explicarán todas las tareas realizadas tanto para configurar el entorno, kernel, variables de entorno, paquetes del sistema, etc... como para instalar correctamente todo el software de EXTOLL para que funcione correctamente mediante el uso de MPI.

A.1. Sistema Operativo y Kernel

Tras varios intentos fallidos intentando instalar el software de EXTOLL sobre el Sistema Operativo RedHat, decidimos configurar nuestro entorno de trabajo de la forma más parecida posible al entorno de trabajo que utilizaban en Alemania, ya que todas las instrucciones de instalación que nos proporcionaban estaban probadas en sus equipos.

Por este motivo instalamos el mismo Sistema Operativo que usan en Alemania, SUSE 10.3, y una versión del Kernel muy parecida (2.6.27.32). Para configurar el Kernel utilizamos el mismo fichero de configuración que usaba la universidad alemana, obteniendo al final una configuración del sistema prácticamente idéntica a la suya.

Comandos básicos de instalación del SO y actualización del Kernel:

```
o Instalacion del SO SUSE 10.3 de forma remota, seguir los pasos
o habituales y en la primera pantalla de Instalación introducir:
UseSSH=1 SSHPassword='El password que quieras'
o Después elegir eth0 y dhcp

o Actualización del Kernel al 2.6.27.32-default
o Nos logamos como root y copiamos kernelGIT.tgz en /usr/src
# cd /usr/src
# tar -xvzf kernelGIT.tgz
# rm linux (simbolic link)
# ln -s linux-2.6.27.y/ linux
# cd /usr/src/linux
o Copiamos el fichero de configuración de los alemanes en este
o directorio renombrándolo a .config y ejecutamos:
# make dep
# make clean
# make bzImage
# make modules
# make modules_install
# cp /usr/src/linux/arch/x86/boot/bzImage /boot/vmlinuz-2.6.27.y
# cp /usr/src/linux/System.map /boot/System.map-2.6.27.y
# mkinitrd -k /boot/vmlinuz-2.6.27.y -i /boot/initrd-2.6.27.y -M\
   /boot/System.map-2.6.27.y
o Actualizamos el fichero /boot/grub/menu.lst para que podamos
o arrancar el sistema con el nuevo Kernel
```

Ejemplo del fichero menu.lst

```
# Modified by YaST2. Last modification on mar dic  9 14:09:32 UTC 2008
default 0
timeout 8
##YaST - generic_mbr
gfxmenu (hd0,5)/boot/message
##YaST - activate

###Don't change this comment - YaST2 identifier: Original name: linux###
title openSUSE 10.3 - 2.6.27.y (Kernel para las tarjetas)
    root (hd0,5)
    kernel /boot/vmlinuz-2.6.27.y root=/dev/sda6 vga=0x317
        resume=/dev/sda6 splash=silent showopts
    initrd /boot/initrd-2.6.27.y

###Don't change this comment - YaST2 identifier: Original name: linux###
title openSUSE 10.3
    root (hd0,5)
    kernel /boot/vmlinuz-2.6.22.5-31-default root=/dev/sda6 vga=0x317
        resume=/dev/sda6 splash=silent showopts
    initrd /boot/initrd-2.6.22.5-31-default

title Red Hat Enterprise Linux Server (2.6.23.8)
        root (hd0,1)
        kernel /boot/vmlinuz-2.6.23.8 ro root=LABEL=/principal rhgb quiet
        initrd /boot/initrd-2.6.23.8.img
title Red Hat Enterprise principal (2.6.18-53.el5)
        root (hd0,1)
        kernel /boot/vmlinuz-2.6.18-53.el5 ro root=LABEL=/principal rhgb quiet
        initrd /boot/initrd-2.6.18-53.el5.img
title Red Hat Enterprise mantenimiento (2.6.18-53.el5)
        root (hd0,0)
        kernel /boot/vmlinuz-2.6.18-53.el5 ro root=LABEL=/ rhgb quiet
        initrd /boot/initrd-2.6.18-53.el5.img
```

A.2. Herramientas de compilación y variables de entorno

Una vez instalado el Sistema Operativo y modificado el Kernel, el siguiente paso consiste en la instalación de las herramientas de compilación necesarias y la configuración de algunas variables de entorno necesarias para una mejor organización de la instalación y gestión de las librerías dinámicas de OpenMPI.

Instalación de las herramientas de compilación:

```
o Primero instalamos gcc, g++ y gfortran utilizando el YAST
o ( gcc y g++ ya estarán )
# yast

o Instalación de texinfo-4.13, autoconf-2.62, automake-1.11 y libtool-2.2.6
o Para cada una de estas aplicaciones, copiamos el directorio que contiene
o la aplicación y accedemos a él, después ejecutamos:
$ ./configure
$ make
# make install
```

Para que nuestro entorno esté más organizado debemos crear las siguientes variables de entorno en nuestro .bash_profile :

- EXTOLL_PATH - Contiene el directorio donde realizaremos la instalación de todo el software perteneciente a EXTOLL y a OpenMPI.
- AMPI_PATH - Contendrá el directorio en el que instalaremos las aplicaciones MPI y los benchmarks de INTEL.
- LD_LIBRARY_PATH - Contendrá el directorio donde incluiremos todas las librerias de OpenMPI.
- Añadiremos a la variable PATH la ruta a los ejecutables de las aplicaciones que vamos a instalar.
- Otra opción que deberemos incluir en nuestro .bash_profile será: ulimit -l 4194304 - que nos permitirá aumentar el límite de MLOCK para no tener problemas posteriores con el uso de EXTOLL.

Si queremos aumentar el límite de MLOCK para todos los usuarios podemos hacerlo modificando el fichero /etc/security/limits.conf, pero para ello necesitaremos permisos de root. Debemos añadir las siguientes líneas:

```
*       soft    memlock 4194304
*       hard    memlock 4194304
```

Ejemplo de configuración del fichero .bash_profile

```
export EXTOLL_PATH=/extoll_stable
export AMPI_PATH=/apli_mpi
export LD_LIBRARY_PATH=/extoll_stable/lib:/extoll_stable/lib/openmpi
export PATH=/sbin:/usr/sbin:/usr/local/sbin:/usr/local/bin:/usr/bin:\
/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/usr/lib64/jvm/jre/bin:\
/usr/lib/mit/bin:/usr/lib/mit/sbin:/extoll_stable/bin:/apli_mpi/bin
ulimit -l 4194304
```

A.3. Instalación del software de EXTOLL

Para que nuestro sistema pueda funcionar con la tarjeta de EXTOLL debemos instalar todos los componentes software descritos anteriormente:

```
o Instalación de extoll_drv, copiamos el directorio y accedemos a él.
$ cd extoll_drv
o Actualizamos la variable EXTOLL_PATH del fichero Makefile.
o Para establecer el número de procesos que queremos ejecutar como máximo
o en cada máquina, debemos editar la linea 16 del fichero extoll_load.sh
o que tenemos en la carpeta extoll_drv.
o Sutituyendo el valor 128 por el valor que queramos. Éste valor dividido 16
o nos da el número de procesos máximo por máquina. Si lo dejamos en 128
o serían 128/16=8 procesos como máximo en cada máquina, como nuestras
o máquinas tienen 16 cores cambiamos el 128 por 256.
$ make
# make install

o Instalación de libvelo, copiamos el directorio y accedemos a él.
$ cd libvelo
$ admin/bootstrap
$ ./configure --prefix=$EXTOLL_PATH
$ make
$ make install
o Ignoramos el error 'missing html files'

o Instalación de librma, copiamos el directorio y accedemos a él.
$ cd librma
$ admin/bootstrap
$ ./configure --prefix=$EXTOLL_PATH
$ make
$ make install
o Ignoramos el error 'missing html files'

o Instalación de velorun, copiamos el directorio y accedemos a él.
$ cd velorun
$ admin/bootstrap
$ ./configure --prefix=$EXTOLL_PATH
$ make
$ make install

o Instalación de velo_np
$ cd velo_np
$ make
# make install
o Ignoramos el error sobre kNPvelo
o copiamos el fichero pexpect.py en $EXTOLL_PATH/bin/
$ cp pexpect.py EXTOLL_PATH/bin/
```

A.3.1. Instalación de OpenMPI y ompi_mtl_extoll

Para que OpenMPI pueda funcionar utilizando EXTOLL debemos instalar la versión 1.3 de OpenMPI de forma normal y posteriormente debemos instalar ompi_mtl_extoll.

Comandos de instalación:

```
o Instalación de OpenMPI
$ cd openmpi-1.3.3
$ ./configure --prefix=$EXTOLL_PATH
$  make all install

o Instalación de ompi_mtl_extoll
$ cd ompi_mtl_extoll
$ admin/bootstrap
$ ./configure --prefix=$EXTOLL_PATH --with-ompi-dir=/extoll_stable\
$ /openmpi-1.3.3/
$ make
$ make install
```

A.3.2. Instalación de ftdi_tools

Para poder reprogramar la FPGA de la tarjeta de EXTOLL debemos instalar las ftdi_tools. Estas herramientas son muy fáciles de instalar y nos permitirán reprogramar la FPGA a través de una conexión Universal Serial Bus (USB).

Comandos para instalar las ftdi_tools

```
o Instalación ftdi_tools
o Primero tenemos que instalar el paquete libusb,
o lo podemos realizar mediante rpm.
o Seguidamente copiamos el directorio en la ubicación que queramos
o y procedemos a instalarlo.
$ cd ftdi_tools
$ admin/bootstrap
$ ./configure --prefix=$EXTOLL_PATH
$ make
$ make install
```

Apéndice B

Configuración del encaminamiento en EXTOLL

En este apéndice se mostrará el funcionamiento del script desarrollado en este Proyecto Final de Carrera para configurar el encaminamiento en los nodos EXTOLL.

B.1. Parámetros del script

Los parámetros necesarios para ejecutar el script son:

- 1 - Tamaño en el eje X de la malla. Posibles valores, 1-8 en caso de que el tamaño en Z sea 1 y 1-4 si el tamaño en Z es >1.
- 2 - Tamaño en el eje Y de la malla. Posibles valores, 1-8 en caso de que el tamaño en Z sea 1 y 1-4 si el tamaño en Z es >1.
- 3 - Tamaño en el eje Z de la malla. Posibles valores 1-4.
- 4 - Nombre del nodo inicial de la malla. Posibles valores jp(1-64). Será el nodo situado más a la izquierda y más al norte de la malla.
- 5 - Comprobación de la malla. Posibles valores 0-1. 0 no se realizará chequeo, 1 la comprobación será realizada. La comprobación consiste en ratificar la conexión desde cada uno de los nodos hasta el resto, por medio del envío de 16 mensajes MPI simultáneos de 2KB.

B.2. Malla 2D

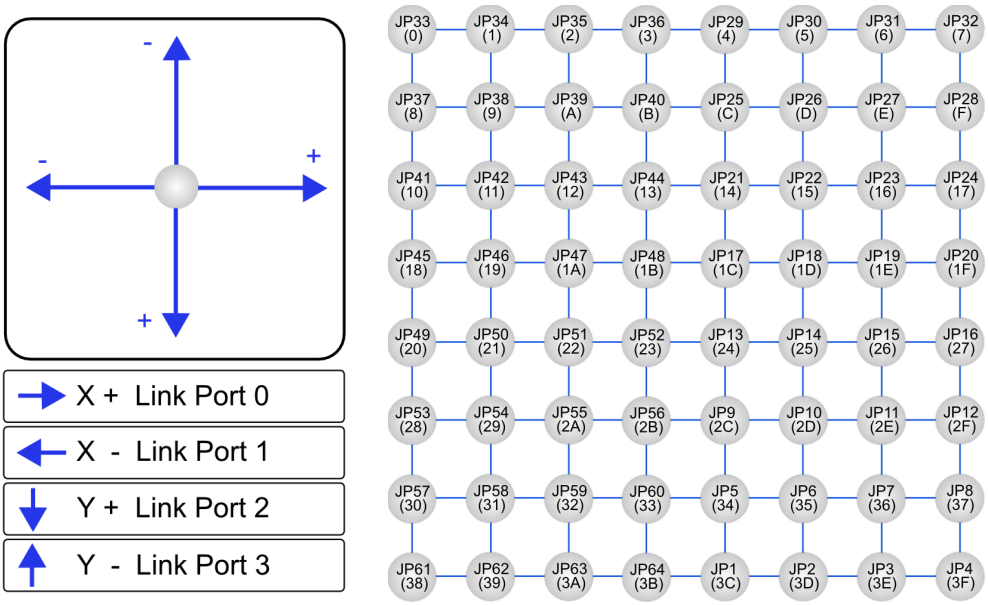

Figura B.1: Imagen base para seleccionar cualquier malla 2D

A partir de la imagen de la Figura B.1 se podrá configurar cualquier malla 2D de tamaño máximo 8x8x1.

Si ejecutamos tres veces seguidas el script de routing de esta forma:

```
[japraga@jp0 ~]$ route/bin/route.sh 4 5 1 jp33 0
[japraga@jp0 ~]$ route/bin/route.sh 4 3 1 jp53 0
[japraga@jp0 ~]$ route/bin/route.sh 4 8 1 jp29 1
```

Se configurarán las mallas 2D de la Figura B.2. Además si nos fijamos en los comandos ejecutados, vemos que las dos primeras mallas simplemente son configuradas sin comprobar sus enlaces, mientras que la última es configurada y comprobada.

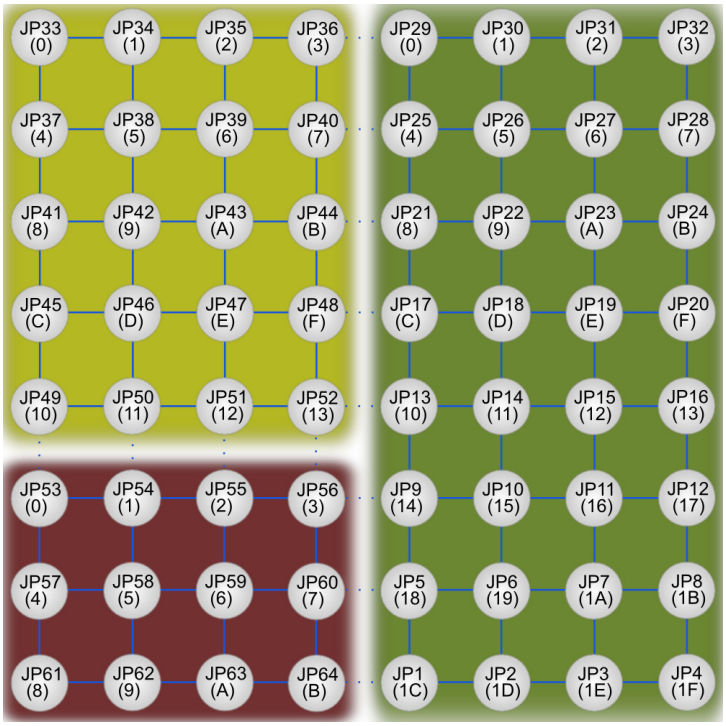

Figura B.2: Configuración de diferentes mallas 2D independientes entre sí

B.3. Malla 3D

Cualquier malla 3D de tamaño máximo 4x4x4 podrá ser configurada mediante nuestro script. Para ello partiremos de la Figura B.3 obtenida a partir de la imagen base 2D tal como se ha visto en el Capítulo 3. En esta figura se muestran los 4 posibles planos XY que pueden formarse al configurar mallas 3D en nuestro prototipo.

Si ejecutamos cuatro veces seguidas el script de routing de esta forma:

```
[japraga@jp0 ~]$ route/bin/route.sh 2 2 2 jp33 0
[japraga@jp0 ~]$ route/bin/route.sh 2 2 2 jp13 0
[japraga@jp0 ~]$ route/bin/route.sh 2 2 4 jp35 1
[japraga@jp0 ~]$ route/bin/route.sh 4 2 4 jp41 1
```

Se configurarán las mallas 3D de la Figura B.4. Obsérvese también que únicamente las dos últimas mallas serán comprobadas.

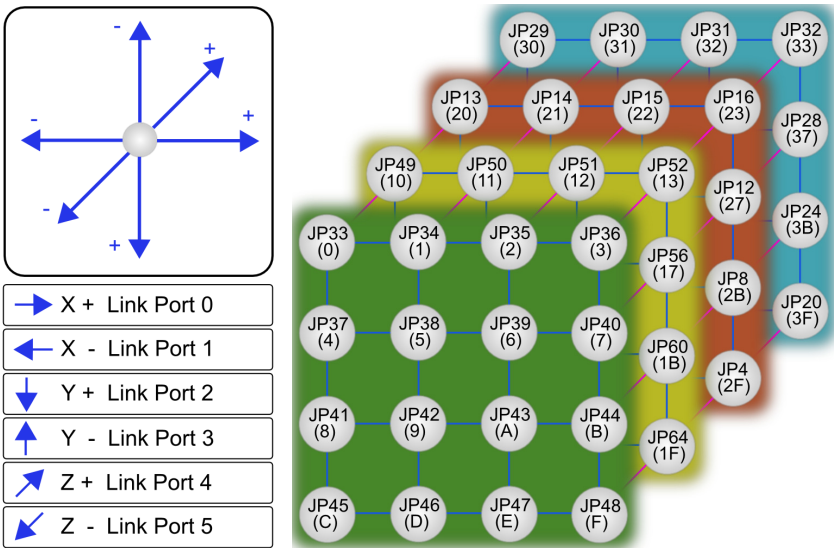

Figura B.3: Imagen base para seleccionar cualquier malla 3D

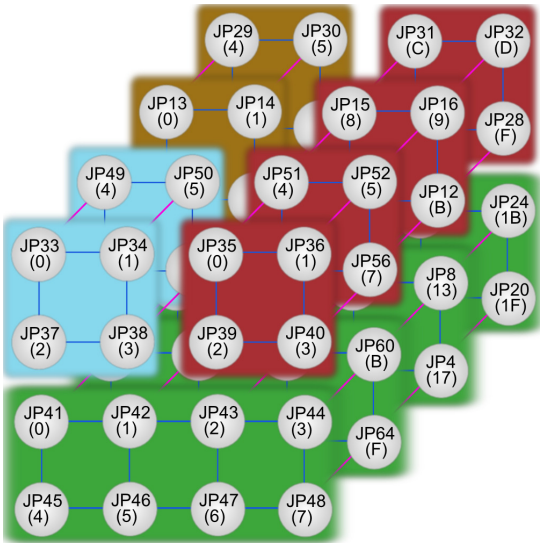

Figura B.4: Configuración de diferentes mallas 3D independientes entre ellas

Apéndice C

Instalación de las aplicaciones paralelas

En este apéndice mostraremos los pasos seguidos para instalar las aplicaciones paralelas que hemos usado en este trabajo para evaluar y comparar el rendimiento del uso de las redes de interconexión EXTOLL y Gigabit Ethernet.

C.1. IMB Test Suite

Para instalar el conjunto de benchmarks MPI de INTEL no es necesaria ninguna instalación previa y los pasos a seguir son:

```
$ wget --no-check-certificate https://software.intel.com/sites/\
$ default/files/managed/7c/17/IMB_4.0.tgz
$ tar xvzf IMB_4.0.tgz
$ cd imb/src/
o Modificamos la variable CC del fichero make_ict para que contenga mpicc
$ make all
o Se crearán tres ejecutables IMB-EXT, IMB-IO y IMB-MPI1.
```

C.2. Instalación de las librerías comunes

La instalación de las aplicaciones paralelas mostrada a continuación necesita en algunos casos de la instalación previa de alguna librería, por eso en este apartado se mostrará la instalación de estas librerías comunes.

Instalación de la librería Fastest Fourier Transform in the West (FFTW):

```
$ wget http://www.fftw.org/fftw-3.3.4.tar.gz
$ tar xvzf fftw-3.3.4.tar.gz
$ cd fftw-3.3.4/
$ ./configure --enable-threads --enable-sse --enable-mpi --enable-float
$ --prefix=(directorio deseado)
$ make
$ make install
```

Instalación de las librerías Basic Linear Algebra Subprograms (BLAS) y Linear Algebra PACKage (LAPACK):

```
$ wget http://www.netlib.org/lapack/lapack-3.5.0.tgz
$ tar xvzf lapack-3.5.0.tgz
$ cd lapack-3.5.0/
$ cp INSTALL/make.inc.gfortran ./make.inc
o Modificar el archivo make.inc para que sea adecuado a nuestro sistema.
o Para ello hay que establecer los siguientes valores:
o FORTRAN       = gfortran -fPIC
o OPTS          = -fomit-frame-pointer -mfpmath=sse -msse3 -O2\
o -falign-loops=32 -fPIC -m64
o NOOPT         = -fomit-frame-pointer -mfpmath=sse -msse3 -m64
o LOADER        = gfortran -fPIC
o LOADOPTS      = $(OPTS)
$ make -j 16 lapacklib blaslib
o Se crearán las librerías liblapack.a y librefblas.a
o Si queremos generar las librerías dinámicas ejecutar:
$ ld -o ./liblapack.so -shared --whole-archive ./liblapack.a
$ ld -o ./librefblas.so -shared --whole-archive ./librefblas.a
```

C.3. CPMD

Instalación CPMD:

```
o Son necesarias las librerías FFTW, BLAS y LAPACK
o Es necesario registro para obtener el código
$ tar xvzf cpmd-v3_17_1.tar.gz
$ cd CPMD
$ ./mkconfig.sh LINUX-x86_64-FEDORA-MPI > Makefile
o Editar el Makefile
$ make clean
$ make -j 16
```

C.4. DL_POLY

Instalación DL_POLY:

```
o Es necesario registro para obtener el código
$ tar xvzf dl_poly_4.05.tar.gz
$ cd dl_poly_4.05/
$ cp build/Makefile_MPI source/Makefile
o Editar source/Makefile Para incluir una nueva plataforma
o #======================================================================
o javi:
o       $(MAKE) LD="/extoll_stable/bin/mpif90 -o" \
o       FC="/extoll_stable/bin/mpif90 -c" \
o       EX=$(EX) BINROOT=$(BINROOT) $(TYPE)
$ cd source
o Cambiar la línea 42 del fichrero set_bounds.f90 para que compile\
o con la versión 4.2.x de gcc !!
o zero_plus  = Nearest( 0.0_wp , +1.0_wp) => zero_plus = 1.0e-16_wp
$ make javi
```

C.5. LAMMPS

Instalación LAMMPS:

```
o Necesita la librería fftw
o Descargar desde http://lammps.sandia.gov/download.html#tar
$ tar xvzf lammps-stable.tar.gz
$ cd lammps-28Jun14/src
o Editar el fichero MAKE/Makefile.openmpi adecuadamente
o (ubicación librería fftw)
$ make -j 16 openmpi
```

C.6. GROMACS

Instalación GROMACS:

```
o Necesita la librería fftw
$ wget ftp://ftp.gromacs.org/pub/gromacs/gromacs-4.6.5.tar.gz
$ tar xvzf gromacs-4.6.5.tar.gz
$ cd gromacs-4.6.5
$ mkdir build
$ cd build
$ export CMAKE_PREFIX_PATH=$FFTW_SINGLE_PATH
o Asegurate de tener la variable de entorno LD_LIBRARY_PATH
o correctamente configurada
$ cmake .. -DGMX_GPU=OFF -DGMX_MPI=ON\
$ -DCMAKE_INSTALL_PREFIX=(Path deseado)
$ make
# make install
o Elimina el rpath si lo crees conveniente
# chrpath -d $AMPI_PATH/GROMACS/4.6.5/bin/mdrun_mpi
```

Bibliografía

[1] Berkeley Unified Parallel C, último acceso 2014. http://upc.lbl.gov/.

[2] Biochemical Dynamics, GROMACS Site, último acceso 2014. http://www.gromacs.org/.

[3] CPMD Test Suite, último acceso 2014. http://cpmd.org/download.

[4] Gaussianos, último acceso 2014. http://gaussianos.com/benoit-mandelbrot-el-matematico-que-amplio-el-concepto-de-geometria/.

[5] Intel MPI Benchmarks, último acceso 2014. https://software.intel.com/en-us/articles/intel-mpi-benchmarks.

[6] Mellanox, último acceso 2014. http://www.mellanox.com/.

[7] Message Passing Interface, último acceso 2014. http://www.mcs.anl.gov/research/projects/mpi/.

[8] Molecular Dynamics Simulators, LAMMPS Site, último acceso 2014. http://lammps.sandia.gov/.

[9] Molecular Simulation Package, DL_POLY Site, último acceso 2014. http://www.stfc.ac.uk/SCD/research/app/ccg/software/DL_POLY/44516.aspx.

[10] OpenMPI, último acceso 2014. http://www.open-mpi.org/.

[11] Parallelized plane wave / pseudopotential implementation of Density Functional Theory, CPMD Site, último acceso 2014. http://www.cpmd.org/.

[12] TOP500 supercomputer, último acceso 2014. http://www.top500.org.

[13] Web site for Wilkinson and Allen's text on parallel programming, último acceso 2014. http://coitweb.uncc.edu/~abw/parallel/par_prog/.

[14] D. Bonachea and J. Jeong. GASNet: A portable high-performance communication layer for global address-space languages. In *Parallel Computer Architecture Project*, 2002.

[15] H. Fröning, M. Nüssle, H. Litz, and U. Brüning. A case for FPGA based accelerated communication. In *Ninth International Conference on Networks (ICN)*, April 2010.

[16] H.-W. Jin, S. Sur, L. Chai, and D. Panda. LiMIC: support for high-performance MPI intra-node communication on linux clusters. In *34th International Conference on Parallel Processing*, June 2005.

[17] K. Kujat, F. Silla, H. Fröning, and J. Duato. The new extoll conduit for the gasnet networking layer. In *Actas de las XXIII Jornadas de Paralelismo*, pages 340 – 345, september 2012.

[18] H. Litz, H. Fröning, M. Nüssle, and U. Brüning. VELO: A novel communication engine for ultra-low latency message transfers. In *37th International Conference on Parallel Processing*, Sept. 2008.

[19] M. Nüssle, M. Scherer, and U. Brüning. A resource optimized remote-memory-access architecture for low-latency communication. In *38th International Conference on Parallel Processing*, Sept. 2009.

[20] J. Prades, F. Silla, J. Duato, H. Fröning, and M. Nüssle. A new end-to-end flow-control mechanism for high performance computing clusters. In *CLUSTER*, pages 320–328. IEEE, 2012.

[21] I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove. Dl_poly_3: new dimensions in molecular dynamics simulations via massive parallelism. *J. Mater. Chem.*, 16:1911–1918, 2006.

[22] B. Wilkinson and M. Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.