The New EXTOLL Conduit for the GASNet Networking Layer

Knut Kujat, Federico Silla, José Duato[1] y Holger Fröning[2]

Resumen— **Los lenguajes PGAS han demostrado ser una forma intuitiva de programación paralela. Con GASNet, una capa de red independiente del lenguaje, preocupaciones por la compatibilidad ya no son ningún problema, ya que GASNet está diseñado para ser fácil y rápidamente adaptado a una nueva arquitectura de interconexión. EXTOLL es una nueva arquitectura de interconexión que es capaz de transmitir mensajes pequeños eficientemente, con su motor de comunicaciones VELO, y mensajes con grandes cantidades de datos, con su motor de comunicación RMA.**

En este artículo explicaremos nuestro conduit de EXTOLL para GASNet y las decisiones tomadas a la hora de implementar el GASNet Core API. Además, las pruebas realizadas con nuestro cluster de 64 nodos mostrarán un mayor rendimiento de nuestro conduit sobre MPI, también implementado para EXTOLL, y cómo nuestro conduit es comparable con una red InfiniBand.

Palabras clave— **EXTOLL, GASNet, PGAS, VELO, RMA, Conduit**

I. Introduction

UN estudio de la lista de los TOP500 [1] del Noviembre de 2011 revela que el 82% de todos las supercomputadoras son clusters. Todos estos clusters dependen de lenguajes de programación paralela, que a su vez son capaces de manejar de forma eficiente sistemas de memoria distribuida, para poder resolver todos los problemas para los cuales fueron construidos. MPI (Message Passing Interface [2] fue introducido por primera vez en 1994 y es, hoy en día, el lenguaje de programación paralela predominante. MPI sigue el paradigma de paso de mensajes, donde la comunicación entre los procesos de una aplicación paralela es explícitamente tratada por el programador en el código. Aunque este enfoque ha demostrado ser eficiente y efectivo para sistemas a gran escala, deja el grueso del trabajo en manos del programador, haciendo más interesante el uso de lenguajes de programación paralela que se aproximan más al enfoque de memoria compartida.

Contrario al enfoque de paso de mensajes proporcionado por MPI, están los lenguajes Partitioned Global Address Space (PGAS), como el Unified Parallel C (UPC) [3], Co Array Fortran (CAF) [4] o Titanium [5], por mencionar solo unos pocos, son lenguajes de programación paralela que ofrecen al programador un espacio de direcciones común para los procesos de una aplicación paralela para conseguir una programación lo más sencilla e intuitiva posible. Además, los lenguajes más recientes, como UPC, han demostrado ser aun más eficientes y conseguir más rendimiento que MPI debido a la menor sobrecarga software y al uso de operaciones unilaterales que requieren menos recursos que las operaciones bilaterales de MPI [8]. Por estas razones, los lenguajes PGAS aparecen como una sólida alternativa a la bien establecida opción de MPI, que en este momento es el estándar de-facto para programar aplicaciones de cómputo a gran escala.

GASNet (Global Adress Space Networking) [9][10], desarrollado por Lawrence Berkeley National Laboratory (LNBL) y la University of California, Berkeley, es una capa de interconexión portable e independiente del lenguaje que proporciona soporte para lenguajes PGAS con primitivas de alto rendimiento, a nivel del compilador. GASNet normalmente está siendo usado para rellenar el hueco entre lenguajes como UPC, que proporciona a los programadores una visión de sistema de memoria compartida, y el hardware como los clusters que normalmente tienen un sistema de memoria distribuida. En este sentido, aplicaciones de memoria compartida escritas utilizando el lenguaje UPC se compilarán a código GASNet, que trata de forma directa con la naturaleza no compartida de la memoria distribuida.

La disponibilidad universal de GASNet para todos los sistemas es asegurada por la existencia de una versión UDP/IP (UDP conduit) que permite el uso de GASNet en todos los sistemas que soportan la pila de protocolos TCP/IP, ya que el conduit es completamente compatible con estos protocolos. Desafortunadamente, el conduit UDP añade tanta sobrecarga software que no es una opción viable para aplicaciones orientadas al rendimiento. Únicamente tiene utilidad para probar aplicaciones escritas en un lenguaje PGAS.

Cuando se requiere rendimiento (lo cual es el caso habitual), la portabilidad de GASNet se consigue porque cada arquitectura de red tiene su propio conduit. El esquema de conduits dependientes de la arquitectura es necesario porque, por lo general, las redes de computadores de alto rendimiento son propietarios de cada fabricante. La red InfiniBand [6] es un buen ejemplo de esto. Esta red proporciona soporte completo para la pila TCP/IP. Por lo tanto, y como GASNet ofrece soporte para UDP, es tentador pensar que no hay necesidad para un conduit basado en operaciones nativas de InfiniBand como Verbs o RDMA. Sin embargo, como se puede ver en la Figura 1, el conduit de UDP tiene una sobrecarga enorme con respecto al conduit IBV nativo de InfiniBand. Aunque ambos conduit se han probado sobre la misma red InfiniBand de 40 Gbps el conduit nativo presenta un rendimiento de 2 a 3 ordenes de magnitud mejor que el conduit de UDP. Este simple experimento prueba la necesidad de un conduit especifico

[1]Departament d'Informàtica de Sistemes i Computadors, Universitat Politècnica de València, e-mail: knuku@gap.upv.es, {fsilla,jduato}@disca.upv.es

[2]Computer Architecture Group, University of Heidelberg, e-mail: froening@uni-hd.de

para cada arquitectura de red de interconexión.

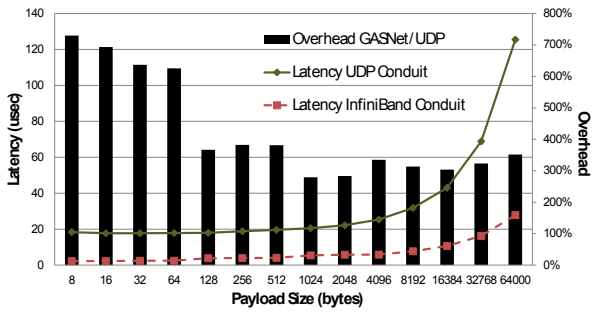


Fig. 1. Sobrecarga de latencia entre el conduit IBV y el conduit UDP

En este artículo presentamos la adaptación de GASNet para la nueva arquitectura de red de interconexión llamada EXTOLL [7]. EXTOLL es una tecnología de red emergente que apunta hacia el mercado de computación de alto rendimiento (HPC) y que proporciona una solución de comunicación innovadora que habilita latencias muy bajas y una tasa de mensajes extremadamente alto, similar en rendimiento a la versión más rápida de InfiniBand pero a un coste mucho más bajo, ya que EXTOLL evita el uso de switches costosos económicamente y con alto consumo de energía. De esta forma, EXTOLL puede añadir un avance dramático en rendimiento en clusters HPC.

En las siguientes secciones introduciremos más a fondo GASNet y EXTOLL , explicaremos más en detalle el nuevo conduit de EXTOLL y presentaremos resultados que mostrarán la eficiencia y la poca sobrecarga del conduit. La Sección 2 resumirá las principales características de GASNet y EXTOLL, mientras que las Secciones 3 y 4 resumirán, respectivamente, el nuevo conduit y un riguroso análisis de las prestaciones obtenidas de resultados reales en nuestro cluster EXTOLL de 64 nodos y 1024 cores. Finalmente terminamos concluyendo nuestro trabajo en la Sección 6.

II. Background

Para proporcionar al lector la base teórica mínima para el estudio presentado en este artículo, en esta sección introduciremos tanto la tecnología de GASNet como la de EXTOLL.

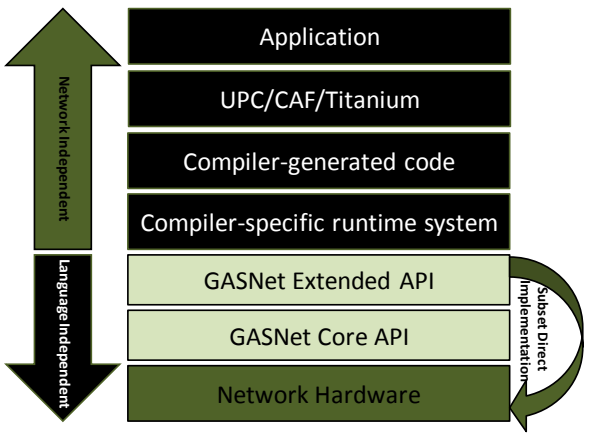


Fig. 2. Diseño de GASNet

A. GASNet

GASNet, como ya se ha mencionado, es una capa de red de comunicación que conecta los diferentes lenguajes PGAS con una variedad de arquitecturas de red. La disponibilidad de GASNet está asegurada debido al uso de un diseño en dos capas, como se puede ver en la Figura 2. La capa Extended API de GASNet proporciona operaciones de memoria a memoria como puts y gets bloqueantes, operaciones de transferencia de registro de memoria y una variedad de operaciones de sincronización como por ejemplo barreras de dos fases.

Por otro lado, el Core API de GASNet, basado en el Active Messages 2 API [11], implementa el esquema básico de paso de mensajes requerido en un sistema distribuido como son los clusters. El Core API de GASNet ofrece un mínimo de operaciones y funciones requeridos para permitir la implementación del Extended API. Consecuentemente, para poder llevar GASNet a una nueva arquitectura de red solo hace falta implementar el Core API, con lo cual el Extended API queda independiente de la tecnología de red que hay por debajo. Sin embargo, para conseguir un mayor rendimiento de la red, un desarrollador de un conduit tiene la posibilidad de eludir el Core API (ver Figura 2) e implementar operaciones del Extended API directamente usando las características que ofrece la arquitectura de red que hay por debajo. Además el Core API y el Extended API no tienen en cuenta particularidades de los lenguajes PGAS. Consecuentemente se puede decir que ambas capas son independientes del lenguaje.

En este artículo nos hemos centrado en implementar únicamente el Core API. Por lo tanto, a continuación explicaremos con más detalle esta capa de GASNet. El Core API tiene que proporcionar las siguientes características para asegurar que el Extended API trabaja de forma correcta:

- Control de trabajos, bootstraping, etc.
- Operaciones de mensajes activos como los requests, replies, y las gestión de los handlers.
- Secciones de no interrupción y locks.

La importancia que tiene un buen desarrollo de la parte de los mensajes activos se puede observar en la Figura 3, que muestra, como ejemplo, el rendimiento obtenido por el benchmark RandomAccess de UPC [15] con dos implementaciones diferentes de los mensajes activos en nuestro conduit EXTOLL. La primera implementación *AM Implementation 1* no tenia el rendimiento esperado. Entonces, tras notables mejoras hemos llegado a la implementación *AM Implementation 2*. Se puede ver claramente que la segunda versión incrementa el rendimiento hasta un 53%.

B. La Arquitectura de Interconexión EXTOLL

EXTOLL es una arquitectura de interconexión que proporciona soporte dedicado a la computación de altas prestaciones.

La arquitectura de EXTOLL, mostrada en la Figura 4, se compone de la interfaz con el host (lado izquierdo de la figura), basado en HyperTransport.

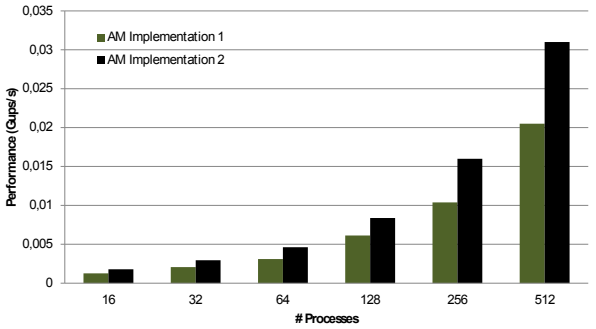


Fig. 3. El RandomAccess de UPC ejecutado con dos implementaciones de los mensajes activos diferentes

En el centro de la figura se puede ver la interfaz con la red y diferentes motores de comunicación, y a la derecha esta la sección de la red con los componentes del switch y seis enlaces con la red.

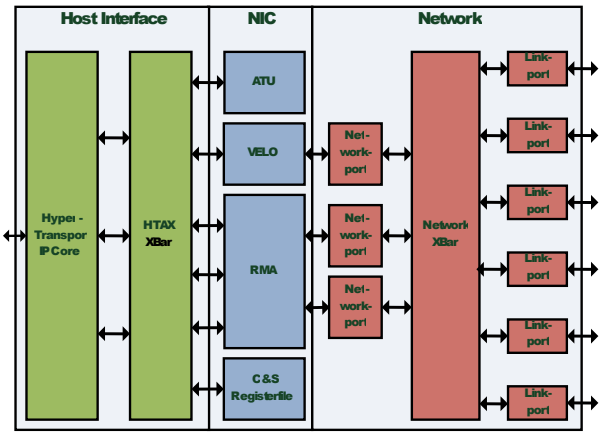


Fig. 4. Diagrama de bloques de la arquitectura de EXTOLL

Como EXTOLL tiene un diseño especifico para un esquema de comunicación de grano fino, incluye un motor de comunicación llamado VELO (*Virtualized Engine for Low Overhead*) [12] que aporta transmisiones optimizadas y con muy bajas latencias para mensajes pequeños. En un paquete VELO se permiten hasta 64 bytes de payload. La recepción se realiza escribiendo los paquetes entrantes en un único buffer circular llamado MAILBOX que esta dividido en slots. Cada slot puede guardar hasta 56 bytes de un paquete VELO más 8 bytes de información de estado.

Además del soporte para mensajes pequeños, EXTOLL trata de forma eficiente transferencias con grandes cantidades de datos mediante el motor de comunicaciones RMA (*Remote Memory Access*) [13] que ofrece operaciones como puts y gets.

III. El Nuevo Conduit de EXTOLL

Hemos explicado en una sección anterior que solo nos hemos centrado en la implementación del Core API de GASNet para obtener una versión de GASNet funcional. El conduit nativo de InfiniBand, el conduit IBV, basado en el conduit VAPI, nos sirvió como plantilla y nos aclaró muchas dudas en el desarrollo.

A. Control de Trabajos

El control del trabajos ya viene en gran medida pre-implementado por la propia suite de GASNet. Sin embargo, hacen falta ajustes como la abertura, conexión e intercambio de endpoints tanto para la transferencia de datos con VELO como con RMA. Además hay que llevar a cabo el *pinning* de regiones de memoria para las transferencias usando accesos a memoria remota.

B. Operaciones de Mensajes Activos

Como hemos podido ver en la Figura 3, una correcta implementación de las operaciones de los mensajes activos es crucial para el rendimiento de cualquier conduit nuevo y por lo tanto para cualquier lenguaje PGAS que se ejecutará sobre el nuevo conduit.

Los mensajes en GASNet tiene tres secciones diferentes. La primera sección es la cabecera de GASNet (8 bytes) indicando nos el tipo de mensaje entrante (request o reply), cuántos argumentos llegarán, el identificador del handler y la longitud del payload de datos. Tras la cabecera está la sección de los argumentos. Se pueden enviar hasta 16 argumentos con una longitud de 4 bytes cada uno. La última sección contiene el payload de datos, en caso de que exista.

El envío de mensajes en GASNet se categoriza, según el tamaño del payload de datos, en tres categorias AM_Short, AM_Medium y AM_Long. Tanto los AM_Short, que no llevan datos, como los AM_Medium, que llevan hasta 8192 bytes en datos, son enviados basándose completamente en el uso de VELO tal y como se muestra en la Figura 5. Sin embargo, también las dos primeras secciones de los AM_Long son enviados usando VELO pero la parte del payload de los datos se envía usando RMA como se puede observar en la Figura 6.

Las primeras dos secciones, la cabecera y los argumentos, de los tres diferentes tipos de mensajes, son enviados usando la función *gasnetc_velo_am_header_send()* que como indica su nombre hace uso del motor de comunicación VELO. Esta función envía según la cantidad de argumentos dos paquetes VELO al destino. Si tras enviar todos los argumentos queda todavía espacio en el último paquete a enviar, que puede ser el primero a enviar, se rellenará este paquete con los primeros bytes del payload de datos en caso de que se trate de un AM_Medium. Los demás datos en un mensaje AM_Medium son enviados en paquetes VELO de solo datos y usando la función *gasnetc_velo_msg_send()*.

La recepción de todos los tipos de mensajes se realiza en la función *AM_Poll()* que consulta (*polling*) el ya mencionado mailbox para ver si existen paquetes nuevos de VELO. En caso positivo, comprueba por el tipo del mensaje y sección dentro del mensaje al cual pertenece el paquete. Si el paquete es de una de las dos primeras secciones de cualquier tipo de mensaje se termina lo usando la función *gasnetc_am_header_recv()*. En caso de que se trate de un paquete VELO de datos se recibe llamando a la función *gasnetc_am_data_recv()*. La terminación

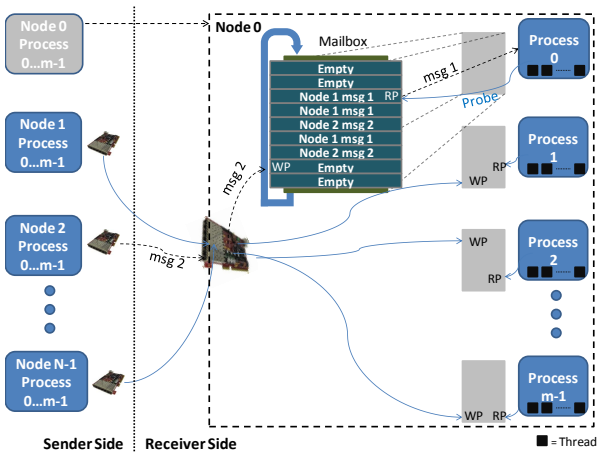


Fig. 5. Recepción de paquetes con VELO

de transferencias RMA se detecta consultando una región de memoria dedicada que contendrá una notificación cuando se ha completado una transferencia.

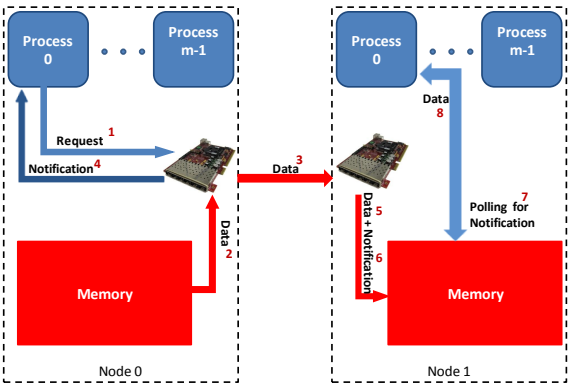


Fig. 6. Transferencias usando RMA

Finalmente, queda por remarcar que tras la recepción de cualquier mensaje se llama al handler específico de cada mensaje, indicado en la cabecera del propio mensaje. En el código del handler cabe la posibilidad de enviar una respuesta al nodo origen del mensaje que llamó al handler.

C. Ejecución Thread-Safe

Todos los procesos en GASNet pueden instanciar diferentes hilos de ejecución (threads) para paralelizar aun más el trabajo a realizar. Por tanto, ciertas secciones tienen que estar protegidas contra accesos concurrentes, como por ejemplo las lecturas y escrituras a la tarjeta EXTOLL. Para hacer el EXTOLL conduit *thread-safe* hemos empleado las funciones *AMLOCK()* y *AMUNLOCK*. Ambas funciones vienen ya dadas por la suite de GASNet y son esencialmente redefiniciones de las operaciones estándar de los POSIX threads adaptados a las necesidades de GASNet.

IV. Análisis de rendimiento

Hemos comprobado el correcto funcionamiento de nuestro EXTOLL conduit usando las pruebas incluidas en la suite de GASNet. Estas pruebas están

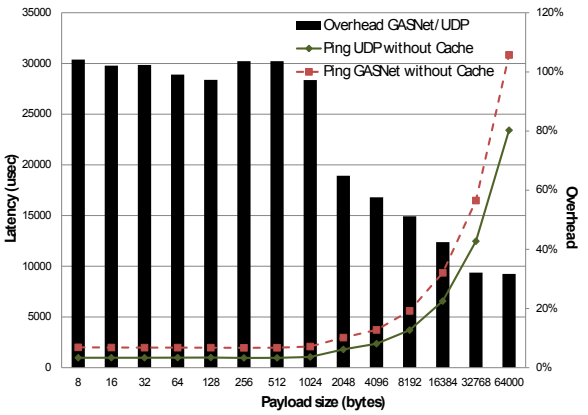


Fig. 8. Ping-Pong con cachés deshabilidados

diseñadas para detectar fallos en nuevos conduits y para facilitar la corrección de esos fallos. No obstante, para mostrar que el nuevo conduit no solo trabaja de forma correcta sino que también tiene un rendimiento prometedor, hemos ejecutado una variedad de pruebas y también un primer benchmark denominado RandomAccess [15]. Hemos comparado los resultados con MPI, el conduit de UDP y el conduit IBV. Todos las pruebas fueron ejecutadas en un sistema real evitando el uso de simuladores.

A. El Cluster y las Pruebas

En nuestro laboratorio tenemos un cluster EXTOLL que consiste en 64 nodos con un total de 1024 núcleos y 1 TB de memoria RAM. Cada nodo tiene las siguientes características:

- 4 procesadores AMD Opteron 8350 con 4 núcleos, trabajando a 2GHz
- 16 GByte de memoria RAM DDR 2
- Tarjeta de red EXTOLL

La tarjeta EXTOLL se ha implementado usando una FPGA Virtex 4 trabajando a 156MHz. Cada enlace proporciona un ancho de banda de 625 MB/s. La topologia usada en la red EXTOLL usualmente es una malla 3D (ver Figura 7(a)) por razones de rendimiento. Sin embargo en algunas pruebas en concreto, y para poder comparar resultados, la red estará configurada en una malla 2D (ver Figura 7(b)).

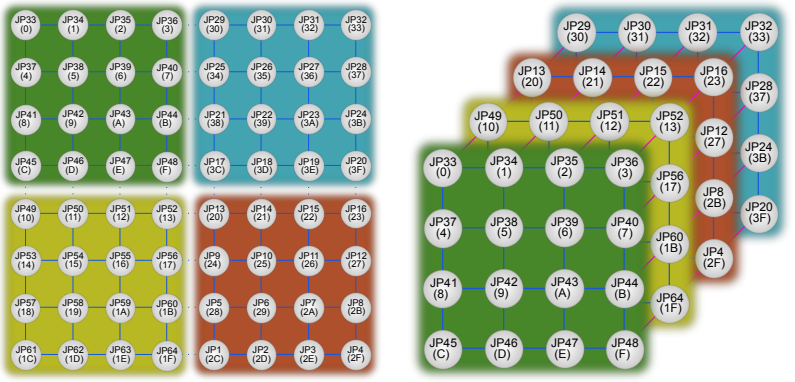
Para las pruebas con InfiniBand hemos usado 2 máquinas con las siguientes características

- 2 procesadores Intel XEON L5530 con 4 núcleos cada uno trabajando a 2.4 GHz
- 24 GBytes de memoria RAM DDR 3
- 1 tarjeta InfiniBand ConnectX-2 EN funcionando a 40 Gbps

Ambas maquinas están conectadas vía un switch InfiniBand MTS3600.

B. Sobrecarga de Latencia

Medir la sobrecarga de latencia es un manera intuitiva de demostrar si la implementación de un conduit es suficientemente eficiente de forma que no tiene un impacto en el rendimiento de GASNet. La prueba elegida para medir la sobrecarga de latencia ha sido un simple programa Ping-Pong.



(a) Malla 2D (b) Malla 3D

Fig. 7. Malla 2D y 3D

En la Figura 8 se pueden observar los resultados de una prueba que mediante la deshabilitación de los cachés de los procesadores da más importancia a la parte software en la ejecución del Ping-Pong. La prueba fue ejecutado usando la versión del Ping-Pong en C usando el protocolo UDP y la versión GASNet con el conduit de UDP. Podemos observar cómo un conduit puede llegar a tener una sobrecarga del 100%.

El programa Ping-Pong en GASNet funciona con todos los conduit de nuestras plataformas (UDP, EXTOLL y InfiniBand). Además hemos implementado la prueba Ping-Pong usando directamente VELO/RMA para poder comparar resultados y calcular la sobrecarga con respecto al conduit de EXTOLL. Para comparar la sobrecarga con el conduit de InfiniBand hemos usado una versión de Ping-Pong implementada con InfiniBand Verbs que se puede obtener de aquí [14]. También hemos implementado nuestra prueba en MPI para poder comparar resultados sobre la misma arquitectura de red pero con diferentes paradigmas.

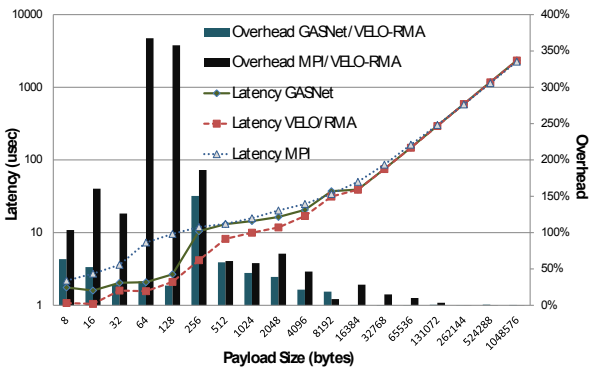


Fig. 9. Sobrecarga de latencia de EXTOLL usando Ping-Pong

La prueba de la Figura 9 muestra las latencias del Ping-Pong obtenidas para las versiones VELO/RMA, MPI, conduit de EXTOLL. Además la figura muestra la sobrecarga (overhead) entre VELO/RMA y MPI y VELO/RMA y EXTOLL tomando VELO/RMA como referencia. Se puede observar que MPI tiene una enorme sobrecarga con respecto a VELO/RMA mientras que el conduit de EXTOLL muestra una mejora en la directa comparación con MPI.

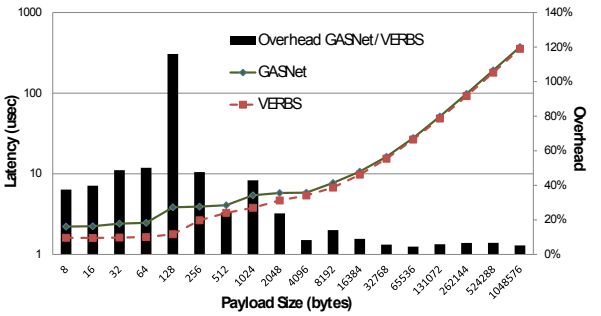


Fig. 10. Sobrecarga de latencia de InfiniBand usando Ping-Pong

La sobrecarga del conduit de InfiniBand con respecto al Ping-Pong implementado directamente con Verbs se puede estudiar en la Figura 10.

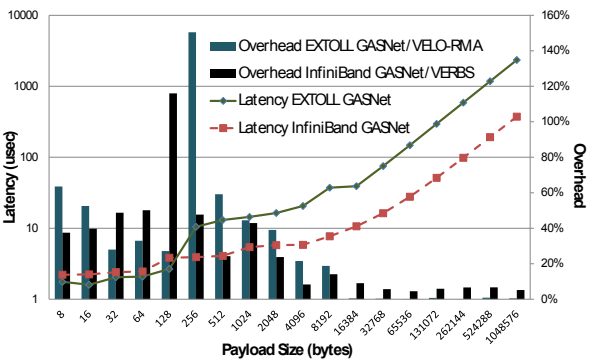


Fig. 11. Comparando sobrecarga entre el conduit de EXTOLL y el conduit InfiniBand

La Figura 11 presenta una comparación directa de las sobrecargas obtenidas con EXTOLL y Infiniband con sus respectivos conduits.

C. Impacto en la Tasa de Mensajes

Además de las pruebas de la sobrecarga de latencia hemos ejecutado el benchmark RandomAccess [15] para ver el impacto sobre la tasa de mensajes enviadas comparando la versión en VELO/RMA con la versión de GASNet sobre nuestro conduit de EXTOLL y la versión en MPI. La pruebas se han ejecutado usando la topologia de la malla 3D.

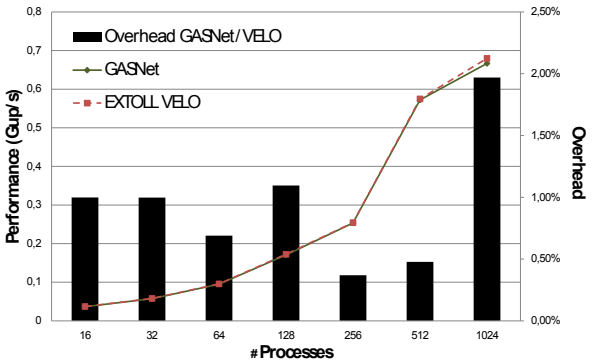


Fig. 12. Comparando sobrecarga entre el RandomAccess usando el conduit de EXTOLL y la implementación del RandomAccess usando VELO

La Figura 12 muestra solo una diferencia marginal entre VELO/RAM y el conduit de EXTOLL.

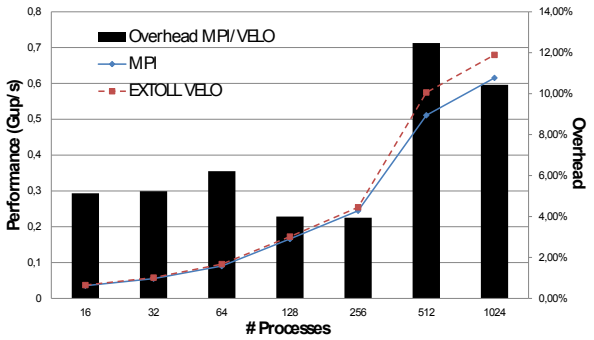


Fig. 13. Comparando sobrecarga entre MPI y VELO usando RandomAccess

La Figura 13 presenta los resultados de MPI con respecto a VELO/RMA y la Figura 14 revela que MPI muestra una notable sobrecarga en comparación directa con nuestro conduit.

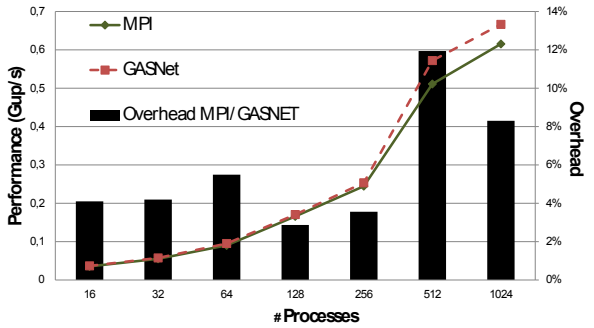


Fig. 14. Comparación directa entre EXTOLL conduit vs MPI usando RandomAccess

El salto en rendimiento que se ha podido observar en las pruebas de RandomAccess en el paso de 256 a 512 procesos ya se ha notado en [16] y tiene como causa la topologia de red sobre la cual se ejecuta la prueba. El salto desaparece para una topologia diferente como se puede ver en la Figura 15.

V. Conclusiones

En este artículo hemos introducido nuestro conduti GASNet de EXTOLL. Hemos repasado brevemente el diseño modular de EXTOLL y sus motores

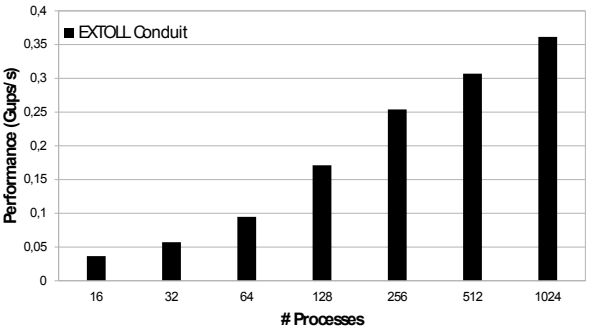


Fig. 15. RandomAccess ejecutado en una red con topologia 2D

de comunicación VELO y RMA. La parte más importante de Core API, los mensajes activos, ha sido explicado en detalle y también se ha visto cómo una pobre implementación de la interfaz de los mensajes activos perjudica de forma considerable el rendimiento de GASNet y de los lenguajes PGAS que usan GASNet como base. La variedad de experimentos ejecutados en las diferentes plataformas, diferentes APIs (UDP,Verbs,VELO/RMA), GASNet y MPI, nos ha permitido realizar un análisis completo de la sobrecarga de la latencia y de la tasa de mensajes.Para completar el análisis hemos hecho comparaciones con una red InfiniBand 8 veces más rápida que nuestra red EXTOLL y se ha podido observar que ambas tecnologías son comparables.

Agradecimientos

Los autores queremos agradecer el trabajo invertido en el GASNet conduit de EXTOLL por Martin Scherer y Dr. Mondrian Nüssle.

Este trabajo fue financiado por Spanish MICINN, Plan E funds, under Grant TIN2009-14475-C04-01.

Referencias

[1] Top 500 homepage, *http://www.top500.org*
[2] Message Passing Interface hompage, *http://www.mcs.anl.gov/research/projects/mpi/*
[3] Berkeley Unified Parallel C homepage, *http://upc.lbl.gov/*
[4] CO Array Fortran homepage, *http://www.co-array.org/*
[5] Titanium homepage, *http://titanium.cs.berkeley.edu/*
[6] Mellanox homepage, *http://www.mellanox.com/*
[7] H. Fröning, et al., *A Case for FPGA based Accelerated Communication The 9th International Conference on Networks* ICN 2010
[8] C. Bell, et al., *Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap*, IPDPS 2006.
[9] D. Bonachea, J. Jeong, *GASNet: A Portable High-Performance Communication Layer for Global Address-Space Languages*, Parallel Computer Architecture Project, Spring 2002.
[10] GASNet homepage, *http://gasnet.cs.berkeley.edu/*
[11] Active Messages Specifications, *http://now.cs.berkeley.edu/AM/active_messages.html*
[12] H. Litz, et al. *A novel communication engine for ultra-low latency message transfers.*, ICPP 2008.
[13] M. Nüssle, et al., *A resource optimized remote-memory-access architecture for low-latency communication.*, ICPP 2009.
[14] http://www.openfabrics.org/downloads/OFED/
[15] http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/
[16] V. Aggarwal, et al. , *HPCC RandomAccess Benchmark for Next Generation Supercomputers*, IPDPS 2009.