UNIVERSITAT JAUME I
ESCUELA SUPERIOR DE TECNOLOGÍA
Y CIENCIAS EXPERIMENTALES

DISEÑO DE UN SISTEMA DE COMUNICACIONES PARA VIRTUALIZACIÓN REMOTA DE ACELERADORES GRÁFICOS SOBRE SISTEMAS HETEROGÉNEOS

CASTELLÓN DE LA PLANA, OCTUBRE DE 2015

PRESENTADO POR: VICENTE R. ROCA SANZ
DIRECTORES: DR. RAFAEL MAYO GUAL
DR. ENRIQUE S. QUINTANA ORTÍ

UNIVERSITAT JAUME I
ESCUELA SUPERIOR DE TECNOLOGÍA
Y CIENCIAS EXPERIMENTALES

DISEÑO DE UN SISTEMA DE COMUNICACIONES PARA VIRTUALIZACIÓN REMOTA DE ACELERADORES GRÁFICOS SOBRE SISTEMAS HETEROGÉNEOS

VICENTE R. ROCA SANZ

A mi esposa, por sacrificar su tiempo
para que yo dispusiera del mío,
y a mis hijos, porque son, han sido y serán
una fuente inagotable de inspiración.

Resumen

El consumo de energía es una de las principales preocupaciones en el diseño de cualquier sistema de HPC y ha sido recientemente reconocido como uno de los grandes retos para alcanzar el siguiente hito en el rendimiento de los supercomputadores: un EXAFLOPS. Para lograr este ambicioso objetivo, es necesario diseñar supercomputadores cada vez más eficientes desde el punto de vista energético, sin perder de vista el rendimiento.

En este contexto, la incorporación de los aceleradores gráficos a los sistemas HPC actuales ha dado lugar a clústeres de máquinas con varios núcleos donde cada nodo está equipado con su propio acelerador. En principio, esto ha supuesto un aumento de la eficiencia energética de estas configuraciones. Sin embargo, los aceleradores pueden permanecer inactivos gran parte del tiempo, durante el cual siguen consumiendo una importante cantidad de energía. Para conseguir un uso más eficiente de las GPUs se han desarrollado varias tecnologías de virtualización de GPUs que permiten ejecutar aplicaciones aceleradas con GPUs accediendo a un acelerador gráfico instalado en un nodo remoto. En la actualidad, la solución más destacada por su robustez, flexibilidad y eficiencia es rCUDA.

Otra de las estrategias para aumentar la eficiencia energética de los clústeres consiste en reemplazar los nodos que incluyen procesadores de propósito general, con un elevado consumo energético, por un número mayor de plataformas con núcleos de menor capacidad de cálculo, pero bajo consumo de potencia eléctrica. Ahora bien, estas configuraciones incrementan el tiempo de ejecución de las aplicaciones de HPC, lo que a larga puede redundar en un mayor consumo de energía.

Este trabajo de investigación aborda el diseño, implementación y evaluación de un sistema de comunicaciones para la virtualización remota de GPUs basado en rCUDA, utilizando redes de alto rendimiento sobre sistemas heterogéneos. En concreto, las propuestas desarrolladas en esta tesis permiten aprovechar las posibilidades de ahorro energético que pueden conseguirse al aplicar la virtualización de GPUs en un clúster heterogéneo que cuenta con nodos basados en procesadores propósito general, plataformas multinúcleo de bajo consumo y arquitecturas híbridas (CPU-GPU) interconectadas por redes de alto rendimiento que soportan

el protocolo RDMA. La evaluación experimental del rendimiento y del consumo energético se efectúa en base a un conjunto de aplicaciones aceleradas con GPUs remotas. El marco de trabajo contempla varias configuraciones representativas de los futuros sistemas de HPC, caracterizados por arquitecturas heterogéneas dirigidas a aumentar la potencia de cálculo teniendo en cuenta la eficiencia energética. Los resultados obtenidos demuestran el potencial de las propuestas desarrolladas en este trabajo para incrementar la eficiencia energética de la solución de virtualización de rCUDA.

Agradecimientos

El ganador del Premio Nobel de Literatura de 1962, John Steinbeck, dijo: "*Las correcciones hechas durante el proceso de creación son, por lo general, excusas para no seguir adelante*". He de reconocer que yo he sido capaz de reunir más de un millón de excusas para no seguir adelante con este trabajo. Sin embargo, gracias al apoyo recibido de muchísima gente, que durante más de seis años estuvieron presentes en mi vida compartiendo alegrías y desventuras, he podido continuar y finalizar lo que entonces empecé. A todos ellos les debo una parte muy importante de este trabajo, y espero poder hacer justicia al intentar transmitir lo que significaron para mí durante este tiempo.

En primer lugar, mi más profundo agradecimiento a mis directores, Rafael Mayo y Enrique S. Quintana, por la confianza que depositaron en mí al aceptarme para realizar esta tesis doctoral bajo su dirección, y porque, a pesar de sus muchas otras ocupaciones y dificultades, se comprometieron y trabajaron intensamente para ayudarme a completar este trabajo, el cual no se puede concebir sin su siempre oportuna participación.

Deseo hacer extensiva mi gratitud a todos los miembros del grupo HPCA, y de manera destacada a Adrián, Sergio, Héctor y Sandra, para quienes sólo tengo palabras de agradecimiento por su compañerismo y ayuda desinteresada.

No puedo olvidar en estos agradecimientos a todos mis compañeros de la Universitat Jaume I y del Departamento de Ingeniería y Ciencia de los Computadores con quienes tengo el placer de trabajar, y especialmente a Germán F., Mar, Asun, José I., Germán L., Juan Carlos F., Pablo B., David, Nacho, Conchi, Tamara y Maribel, que han ayudado con amistad, consejos y buenos ratos a que este trabajo fuese más soportable.

Asimismo, quiero hacer una mención especial a mi compañero y amigo Gustavo por su asesoramiento durante todos estos años de amistad. Con él he compartido conocimientos y experiencias profesionales y personales que me han sido de gran valor.

También deseo mostrar mi agradecimiento a mis amigos, y en particular a mi buen amigo y compañero de fatigas deportivas Vicent. Me siento afortunado de haber podido contar con su apoyo y sus consejos.

En modo alguno habría llegado hasta aquí de no ser por mi esposa, Almudena. Su amor y comprensión me han acompañado en este proceso, y su paciencia y fortaleza me han permitido no sólo trabajar, sino también terminar. Nunca le podré estar suficientemente agradecido.

Un lugar muy especial merecen mis hijos Vicent, Ferran y Aleix, porque han tenido que soportar largas horas sin mi compañía. Sin entender, a su corta edad, el porqué he preferido estar frente a la pantalla del ordenador y no jugando con ellos. Son, sin lugar a dudas, mi referencia para el presente y para el futuro.

Para finalizar, quiero dar las gracias a mi familia política, a mi hermano Raúl y muy especialmente a mi padre, Vicente, a su mujer, Ana, y a mi madre, Montse, quienes participaron, directa e indirectamente, en mi formación. Ellos me han enseñado a encarar las adversidades sin perder nunca la dignidad ni desfallecer, y me han dado todo lo que soy como persona: mis valores y mis principios.

Mi más sincero agradecimiento a todos quienes me habéis acompañado en esta larga jornada que ahora concluye.

Muchas gracias.

Índice general

Índice de figuras

Índice de tablas

Abreviaturas

10GbE Ethernet de 10 Gbit/s.

1GbE Ethernet de 1 Gbit/s.

ALU *Arithmetic Logic Unit*, Unidad Aritmética Lógica.

API *Application Programming Interface*, Interfaz de Programación.

BLAS *Basic Linear Algebra Subprograms*.

BP *Busy Polling*, Consulta Activa.

CA *Channel Adapter*, Adaptador de Canal.

CISC *Complex Instruction Set Computer*.

CPD Centro de Procesamiento de Datos.

CPU *Central Processing Unit*, Unidad Central de Procesamiento.

CQ *Completion Queue*, Cola de Finalización.

CQE *Completion Queue Element*, Elemento de la Cola de Finalización.

CSR *Channel Semantics over RDMA*, Semántica de Canal sobre RDMA.

CUDA *Compute Unified Device Architecture*.

CUDASW++ *CUDA GPU accelerated Smith-Waterman algorithm*.

DCB *IEEE Data Center Bridging*.

DDP *Direct Data Placement Protocol*.

DMA *Direct Memory Access*, Acceso Directo a Memoria.

EDR *Enhanced Data Rate*.

EtS *Energy to Solution*, Energía para la Solución.

FAR *First Approach over RDMA*, Primera Aproximación sobre Redes RDMA.

FPU *Floating-Point Unit*, Unidad de Coma Flotante.

FTTSE Métrica $Function(Time - To - Solution) \cdot Energy$.

GPGPU *General-Purpose computing on Graphics Processing Units*, Cálculo de Propósito General sobre Unidades de Procesamiento Gráfico.

GPU *Graphics Processing Unit*, Unidad de Procesamiento Gráfico.

GPU-BLAST *GPU Basic Local Alignment Search Tool*.

HCA *Host Channel Adapter*.

HLP *Hardware Load at the Plug*, Potencia eléctrica del Equipo.

HPC *High Performance Computing*, Computación de Alto Rendimiento.

HPCC *High Performance Computing Cluster*, Clúster de Computación de Alto Rendimiento.

HPL *High-Performance Linpack benchmark*.

HPL-Fermi *High-Performance Linpack benchmark for Fermi*.

IB InfiniBand.

IBTA *InfiniBand Trade Association*.

IETF *The Internet Engineering Task Force*.

iWARP *Internet Wide Area RDMA Protocol*.

LAMMPS *Large-scale Atomic/Molecular Massively Parallel Simulator*.

LAN *Local Area Network*.

MPA *Marker PDU Aligned*.

MPI *Message Passing Interface*.

MR *Memory Region*, Región de Memoria Registrada.

MSR *Memory Semantics over RDMA*, Semántica de Memoria sobre RDMA.

NCBI-BLAST *National Center for Biotechnology Information Basic Local Alignment Search Tool*.

NetPIPE *Network Protocol Independent Performance Evaluator*.

OFA *OpenFabrics Alliance*.

OFED *OpenFabrics Enterprise Distribution*.

OpenCL *Open Computing Language*.

PCIe *PCI-Express*.

PDU *Power Distribution Unit*, Unidad de Distribución de Potencia.

PEff *Parallel Efficiency*, Eficiencia Paralela.

PFC *Priority Flow Control*.

PpW *Performance per Watt*, Rendimiento por Vatio.

PSU *Power Supply Unit*, Fuente de Alimentación.

QDR *Quad Data Rate*.

QP *Queue Pair*, Par de Colas.

RC *Reliable Connection*, Conexión Fiable.

rCUDA *Remote CUDA*.

RD *Reliable Datagram*, Datagrama Fiable.

RDMA *Remote Direct Memory Access*.

RDMAP *RDMA Protocol*.

RISC *Reduced Instruction Set Computer*.

RNIC *RDMA Network Interface Controller*.

RoCE *RDMA over Converged Ethernet*.

SAI Sistema de Alimentación Ininterrumpida.

SAN *Storage Area Network*.

SIMD *Single Instruction, Multiple Data*.

SO Sistema Operativo.

SoC *System-on-a-Chip*, Sistema Embebido en un Chip.

speed-up Factor de Mejora del Rendimiento.

TCO *Total Cost of Ownership*, Coste Total de Propiedad.

TDP *Thermal Design Power*, Potencia de Diseño Térmico.

TOE *TCP Offload Engine*.

Tpt *Troughput*, Productividad.

TtS *Time to Solution*, Tiempo de Ejecución.

UC *Unreliable Connection*, Conexión no Fiable.

UD *Unreliable Datagram*, Datagrama no Fiable.

UDP *User Datagram Protocol*.

vSMP *Variable Symmetric Multiprocessing*, Multiprocesamiento Simétrico Variable.

WC *Work Completion*, Información de Finalización del Trabajo.

WFE *Wait For Event*, Espera de un Evento.

WQE *Work Queue Element*, Elemento de la Cola de Trabajo.

WR *Work Request*, Solicitud de Trabajo.

Glosario

Adaptador de Canal para Host (*Host Channel Adapter*, HCA) es un dispositivo de red que proporciona a un ordenador conectividad en una infraestructura de red RDMA.

Best effort es el término utilizado en telecomunicaciones para definir la forma de prestar aquellos servicios para los que no se puede garantizar una determinada calidad de servicio (QoS). Esto implica que no existe una preasignación de recursos, ni plazos conocidos, ni garantía de recepción correcta de la información.

Cliente-servidor es un modelo donde unos programas que necesitan servicios (clientes) se comunican a través de una red con los programas que proporcionan los servicios (servidores) para realizar una o varias tareas de forma conjunta. Un cliente hace una petición de un servicio a un servidor, que la procesa y devuelve la respuesta.

Clúster de Computación de Alto Rendimiento (*High Performance Computing Cluster*, HPCC) es un conjunto de ordenadores conectado a través de una red que está diseñado para obtener altas prestaciones en cuanto a capacidad de cálculo.

Cola de Finalización (*Completion Queue*, CQ) es una cola (FIFO) que contiene las estructuras de datos que almacenan la información de finalización de las solicitudes de trabajo (WR).

Conexión Fiable (*Reliable Connection*, RC) es un tipo de servicio de transporte de datos basado en pares de colas que usa un protocolo orientado a la conexión.

Conexión no Fiable (*Unreliable Connection*, UC) es un tipo de servicio de transporte de datos basado en pares de colas. El par de colas ejecuta un protocolo sin conexión de manera que puede existir perdidas de información.

Cálculo de Propósito General sobre Unidades de Procesamiento Gráfico (*General-Purpose computing on Graphics Processing Units*, GPGPU) es la técnica de computación que se basa en la utilización de aceleradores gráficos para realizar operaciones cálculo de propósito general.

Datagrama no Fiable (*Unreliable Datagram*, UD) es el término empleado para nombrar a un tipo de servicio de transporte donde un par de colas puede transmitir mensajes a otro par de la subred. En este tipo de comunicación los mensajes pueden perderse y el orden tampoco está garantizado.

Demonio es el término utilizado para referirse a un proceso que se ejecuta en segundo plano y que realiza una operación especificada en respuesta a ciertos eventos o solicitudes.

Elemento de la Cola de Finalización (*Completion Queue Element*, CQE) hace referencia a una entrada en la cola de finalización (CQ) que contiene la información sobre una solicitud de trabajo completada (WC).

Elemento de la Cola de Trabajo (*Work Queue Element*, WQE) es el término que se utiliza para nombrar a cada una de las entradas de la cola de trabajo (QP).

Enlazador dinámico (*dynamic linker*) es la parte de un sistema operativo que carga y enlaza las bibliotecas necesarias para un ejecutable en tiempo de ejecución.

Extremo remoto describe al extremo más alejado de una comunicación punto a punto.

GPUDirect RDMA es una tecnología introducida en CUDA 5.0 que habilita una ruta directa para el intercambio de datos entre la GPU y un dispositivo de terceros usando los servicios del bus PCI-Express. Entre estos dispositivos se encuentran interfaces de red, dispositivos de adquisición de vídeo, adaptadores de almacenamiento y otras GPUs.

Host es el término que hace referencia al ordenador que tiene conectado un dispositivo compatible con *CUDA* o que puede controlar uno o más adaptadores de red.

Información de Finalización del Trabajo (*Work Completion*, WC) se obtiene como resultado de procesar una solicitud de trabajo (WR).

Interfaz de Programación (*Application Programming Interface*, API) es un conjunto definido de reglas y especificaciones que un programa debe seguir para acceder y hacer uso de los servicios y recursos proporcionados por otro programa o sistema que implementa esa API.

Kernel es el término que usa CUDA para describir a una función que al ejecutarse en la GPU lo hace en *N* hebras distintas en lugar de hacerlo de forma secuencial.

Memoria no paginable es la porción de memoria que no puede ser llevada a disco (*swapping*) lo que permite que ciertos dispositivos PCI-Express puedan solicitar traslados desde y hacia ella sin intervención de la CPU.

Memoria paginable designa a la zona de memoria principal que puede ser copiada en disco, de modo que cuando se necesita esa página, la CPU debe cargarla de nuevo. Esto permite a los programadores utilizar un espacio de direcciones virtual más grande que el realmente disponible en la memoria RAM.

Multi-instancia se refiere a la arquitectura cliente-servidor donde cada cliente tiene su propia instancia servidor asignada.

Método de Consulta Activa (*Busy Polling*, BP) es uno de los mecanismos implementados en los módulos de comunicaciones sobre redes RDMA desarrollados en el presente trabajo para detectar la finalización de las operaciones de comunicación. Realiza una consulta explícita del estado de las colas de finalización a intervalos constantes.

Método de Espera de un Evento (*Wait For Event*, WFE) hace referencia a una de las técnicas implementadas en los módulos de comunicaciones

sobre redes RDMA desarrollados en el presente trabajo para detectar la finalización de las operaciones de comunicación. Emplea llamadas del sistema operativo que bloquean el proceso de comunicación hasta que se produce la notificación de que una operación de comunicación ha finalizado.

Módulo CSR (*Channel Semantics over RDMA*) es el término empleado para referirse al módulo de comunicaciones sobre redes RDMA que sólo utiliza operaciones de semántica de canal de la biblioteca *rdmacm*.

Módulo FAR (*First Approach over RDMA*) es la primera aproximación para el módulo de comunicaciones sobre redes RDMA.

Módulo MSR (*Memory Semantics over RDMA*) se refiere al módulo de comunicaciones sobre redes RDMA que emplea operaciones de semántica de memoria de la biblioteca *rdmacm*.

Par de Colas (*Queue Pair*, QP) es el par formado por la cola de envío y la cola de recepción empaquetadas conjuntamente en un objeto cuya finalidad es la transferencia de datos entre los nodos de una red RDMA.

Pinned memory es el término usado en CUDA para designar a la memoria no paginable.

Pipeline es la técnica de solapar la ejecución de diferentes operaciones al mismo tiempo.

Potencia de Diseño Térmico (*Thermal Design Power*, TDP) es la máxima cantidad de potencia requerida por el sistema de refrigeración de un sistema informático para disipar el calor generado por los elementos de cómputo de un ordenador. Esta métrica es utilizada por los fabricantes de procesadores y aceleradores de cálculo para proporcionar información de su consumo energético.

rdmacm (*RDMA Communication Manager library*) proporciona una interfaz de transporte neutral sobre RDMA para establecer conexiones. Los conceptos de la API se basan en sockets, pero adaptados al par de colas (QP). La biblioteca controla tanto el QP como la gestión de

la comunicación. Funciona conjuntamente con la API definida por la biblioteca `ibverbs` que proporciona las interfaces subyacentes necesarios para enviar y recibir datos.

Región de Memoria (*Memory Region*, MR) sirve para describir a una porción de memoria que se ha registrado para que el adaptador de canal (CA) pueda hacer uso de ella en las transferencias.

Solicitud de Trabajo (*Work Request*, WR) es una petición RDMA que será insertada por un usuario en una cola de trabajo (QP).

Stream es una secuencia de operaciones que se ejecutan en el dispositivo CUDA en el orden en que han sido emitidos por la aplicación. Mientras que se garantiza que las operaciones dentro del *stream* ejecutarán en el orden prescrito, las operaciones en diferentes *stream* pueden ser intercaladas y, cuando sea posible, pueden ejecutadas simultáneamente.

Swapping es una técnica que permite al ordenador ejecutar programas y manipular archivos de datos más grandes que la memoria principal. El sistema operativo copia tantos datos como sea posible en la memoria principal, y deja el resto en el disco. Cuando el sistema operativo necesita los datos desde el disco, intercambia una porción de datos de la memoria principal (llamada página o segmento) por una porción de los datos del disco.

TCP offload engine (*TCP offload engine*, TOE) es una tecnología utilizada en las tarjetas de red para reducir la carga de procesamiento de la pila TCP/IP en la CPU. Se utiliza principalmente en los adaptadores de red para 1 Gigabit Ethernet y 10 Gigabit Ethernet.

TCP/IP es la familia de protocolos utilizada en Internet. Proviene de los nombres de los dos protocolos más importantes de este conjunto de protocolos.

Thread-safe es el calificativo empleado para designar a una porción de código que funciona correctamente durante la ejecución simultánea en múltiples hebras. En particular, debe satisfacer la condición de

que varias hebras puedan acceder a los mismos datos, y de que los datos compartidos sean accedidos por una sola hebra en un momento dado.

Verbs o "verbos" es la descripción abstracta de la funcionalidad de un adaptador de canal para RDMA. El sistema operativo proporciona estas funcionalidades a través de una o más API a las aplicaciones que realizan transferencias de datos utilizando la tecnología RDMA.

Zero-copy describe las operaciones de transferencia en las que la CPU no realiza la tarea de copiar datos de una área de memoria a otra. Esta técnica se utiliza para ahorrar tiempo y memoria a la hora de enviar datos a través de una red.

CAPÍTULO 1

Introducción

1.1. Motivación

Los Centros de Procesamiento de Datos, o CPD, albergan grandes cantidades de equipos informáticos usados para el procesamiento y el almacenamiento de datos, así como dispositivos de red que interconectan estos equipos entre sí y el mundo exterior. Los clústeres son grupos de equipos (denominados nodos) interconectados que habitualmente cooperan para realizar una tarea o resolver un problema relacionado principalmente con el cálculo de alto rendimiento o HPC (*High Performance Computing*), la provisión de servicios o el almacenamiento de datos. En un CPD de tamaño medio puede haber hasta un millar nodos agrupados en diferentes clústeres. Al igual que cualquier dispositivo eléctrico o electrónico, un equipo informático genera calor, y cada nodo puede compararse con un pequeño radiador que disipa una potencia (instantánea) de entre 600 W y 1.000 W. Debido a la gran cantidad de calor producido por todo el hardware en uso, los CPDs deben tener un sistema de refrigeración mucho más potente que los equipos destinados al uso doméstico, lo que supone un consumo adicional de energía considerable. Al mismo tiempo, el número de CPDs ha aumentado significativamente en los últimos años debido a la aparición de nuevos servicios y a la gran demanda de tecnologías capaces procesar un gran volumen de datos.

Este escenario descrito anteriormente se traduce en un incremento notable del consumo energético global de estos centros, alcanzando en los

Estados Unidos (EE.UU.), el país con más centros de este tipo, hasta el 3 % del consumo nacional de energía, y el equivalente a una factura eléctrica de 9.000 millones de dólares al año según las últimas estimaciones realizadas en 2014 por la Agencia de Protección Ambiental de EE.UU. (EPA) [1].

Junto a las implicaciones económicas, la creciente preocupación por el cambio climático, unido a la rapidez y virulencia con la que se están mostrando sus consecuencias en forma de desastres naturales, han hecho aumentar el interés por lo que se conoce como la "Tecnología Verde" o *Green technology*, término general que incluye todos los aspectos de la ecología aplicados a la informática.

Asimismo, uno de los principales desafíos que se presentan para conseguir la capacidad de computación de un EXAFLOPS (10^{18} operaciones en coma flotante por segundo) es aumentar el número de cálculos respecto al consumo de energía [2-6]. Para ello es necesario diseñar supercomputadores cada vez más eficientes desde el punto de vista energético, sin perder de vista el rendimiento [7-12].

Una de las alternativas en esta dirección que ha recibido más atención ha sido el remplazo de los nodos que incluyen procesadores de propósito general, con un elevado consumo energético, por un número mayor de plataformas con núcleos de menor potencia de cálculo pero bajo consumo energético. Mientras que los procesadores presentes en los actuales sistemas de HPC alcanzan en la mayoría de los clústeres un consumo medio de 130 W, los procesadores de bajo consumo tienen unas cifras muy inferiores, que pueden llegar hasta el 2 % de ese valor, como es el caso de los procesadores Intel Atom N2600 y ARM Cortex A9 [13].

El uso de tecnologías basadas en sistemas integrados no es nuevo en el ámbito de HPC. La serie Blue Gene de IBM es un buen ejemplo que demuestra el potencial y la viabilidad de utilizar estas plataformas en los CPDs. Estos sistemas, basados en la arquitectura PowerPC 440 usada en Blue Gene/L, fueron originalmente diseñados para el mercado de dispositivos móviles [14] y ahora se encuentran en 19 de los primeros 25 sistemas de la última lista Green500 [15]. Sin embargo, los procesadores Intel Atom y ARM tiene la ventaja de ser más baratos que los procesadores de Blue Gene, por lo que están presentes en la mayoría dispositivos móviles y sistemas empotrados actuales.

Otra línea de investigación dirigida a mejorar la eficiencia energética

consiste en incorporar unidades de procesamiento gráfico, o GPUs (*Graphics Processing Unit*s), en los nodos para acelerar el cálculo (véase la Figura 1.1). Inicialmente, estos dispositivos fueron diseñados exclusivamente para el procesamiento gráfico, pero con el tiempo se han incorporado al mundo HPC para hacer frente a las tareas de cálculo de propósito general, lo que se conoce como cálculo de propósito general sobre unidades de procesamiento gráfico o GPGPU (*General-Purpose computing on Graphics Processing Units*). Esta nueva tendencia se ha visto favorecida por las facilidades introducidas en la programación de la arquitecturas GPU, con estándares de programación como CUDA (*Compute Unified Device Architecture*) [16] y OpenCL [17], su gran potencia de cálculo (para ciertas aplicaciones), y un precio asequible. De ahí que los HPCCs (*High Performance Computing Cluster*s) estén comenzando a utilizar estos dispositivos para acelerar ciertas porciones de la aplicación que realizan cálculo intensivo. Así, por ejemplo, a fecha de junio de 2015, entre los diez primeros puestos de la lista TOP500 se encuentran dos supercomputadores equipados con GPUs [18].

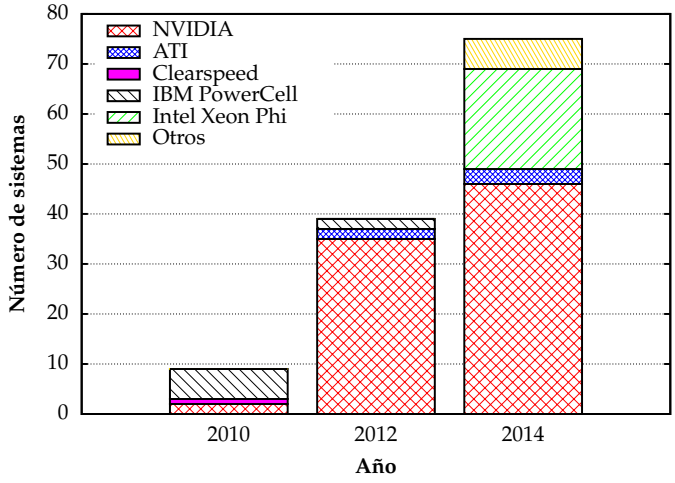


Figura 1.1: Evolución de los aceleradores en la lista TOP500.

En algunos casos, el aprovechamiento de aceleradores gráficos puede llegar a reducir el tiempo de ejecución a una centésima parte del original, si bien depende en gran medida de cada aplicación en particular. No obstante, la reducción del tiempo de ejecución conlleva un ahorro en el consumo de energía del sistema puesto que son dispositivos muy eficientes

energéticamente mientras efectúan los cálculos. Ahora bien, cuando están en reposo presentan un consumo de energía nada despreciable.

Por otro lado, las aplicaciones que se ejecutan en estos clústeres equipados con aceleradores deben cumplir ciertas condiciones para conseguir beneficiarse de la ejecución en las GPUs, algo que no siempre ocurre. Además, a menudo sólo es posible acelerar una pequeña parte de la aplicación. Como resultado, se pueden encontrar clústeres donde todos los nodos están equipados con una GPU, y donde el acelerador permanece inactivo gran parte del tiempo durante el cual sigue consumiendo una importante cantidad de energía. Este "desperdicio" puede llegar a representar más del 30 % del total de la energía consumida por el nodo, lo que tiene un gran impacto tanto en el consumo de energía como en el coste total de propiedad o TCO (*Total Cost of Ownership*) del sistema.

Con el propósito de conseguir un uso más eficiente de estos recursos de aceleración, el grupo Parallel Architectures de la Universitat Politècnica de València y el grupo High Performance Computing and Architectures de la Universitat Jaume I han desarrollado hasta la fecha una iniciativa conjunta conocida como "The Remote CUDA (rCUDA) Project" [19]. El software rCUDA ofrece la posibilidad de ejecutar las diversas partes de una aplicación en los diferentes aceleradores instalados en el clúster, independientemente de la localización física del acelerador gráfico. De este modo, rCUDA es capaz de ofrecer dispositivos de aceleración virtualizados a todos los nodos de un clúster a cambio de añadir una pequeña sobrecarga. El propósito de rCUDA en los HPCC es reducir el número de dispositivos de aceleración, de forma que no es necesario instalar una GPU en cada nodo del clúster, lo que permite incrementar el tiempo de utilización de estos dispositivos. Esto supone un ahorro en los costes de adquisición, de consumo de energía, de refrigeración, de espacio y de mantenimiento.

Resulta obvio que la aceleración de una aplicación con GPUs remotas genera cierto sobrecoste si se compara con la aceleración local, ya que las solicitudes de operación y los datos deben atravesar la red que conecta los nodos cliente, donde se ejecutan las aplicaciones de usuario, y los equipos servidores dotados con GPUs. El software de rCUDA soporta la virtualización a través de redes TCP/IP, lo que permite su uso en la mayoría de las tecnologías de red existentes, y también en infraestructuras InfiniBand.

Por otro lado, son muchas las posibilidades de ahorro que pueden conseguirse al aplicar la virtualización de GPUs en un clúster heterogéneo que cuenta con nodos basados en procesadores propósito general, plataformas multinúcleo de bajo consumo y arquitecturas híbridas (CPU-GPU) interconectadas por redes de alto rendimiento. Sin embargo, la utilización de un sistema de comunicaciones basado en TCP/IP impide que las transferencias de datos a la GPU aprovechen todas las ventajas que ofrecen las tecnologías de red de alto rendimiento, las cuales disponen de operaciones de comunicación específicas con mayor rendimiento. En respuesta a esta carencia el objetivo general de esta tesis es *el diseño, desarrollo y evaluación de un módulo de comunicaciones para la virtualización remota de GPUs utilizando redes de alto rendimiento sobre sistemas heterogéneos*.

En este contexto, se supone que los sistemas basados en arquitecturas de bajo consumo pueden lograr una mayor eficiencia energética que los sistemas con procesadores de propósito general. Ahora bien, los procesadores de bajo consumo Intel Atom y ARM son una tecnología relativamente nueva en el ámbito de HPC, y su eficiencia energética en este entorno es todavía incierta [20-26]. Por esta razón, en línea con el objetivo principal de la tesis y motivado por la necesidad de validar los beneficios obtenidos, se plantea la evaluación del potencial y de los sobrecostes que ofrecen los sistemas heterogéneos a las aplicaciones paralelas científicas aceleradas con GPUs remotas sobre redes de alto rendimiento.

1.2. Objetivos

Como se ha introducido anteriormente, el objetivo fundamental de la tesis es el diseño, desarrollo y evaluación de un módulo de comunicaciones, para la virtualización remota de GPUs, utilizando redes de alto rendimiento sobre sistemas heterogéneos. Este objetivo general se puede descomponer en los siguientes objetivos específicos:

- Diseñar, implementar y evaluar un prototipo de sistemas de comunicaciones con el propósito de explorar el uso de primitivas de comunicación específicas para redes de alto rendimiento que emplean protocolos RDMA (*Remote Direct Memory Access*). La finalidad de este desempeño inicial es ofrecer una mejora en el rendimiento de las

transferencias entre sistemas de bajo consumo y sistemas servidores equipados con GPUs frente a las comunicaciones basadas en TCP/IP.

- A partir de los mecanismos desarrollados en el prototipo inicial, diseñar e implementar dos componentes del sistema de virtualización de GPUs de rCUDA adaptados a cada una de las semánticas de comunicación de las tecnologías RDMA (canal y memoria) para aumentar su rendimiento sobre estas redes. Como parte de este objetivo también se incluye la elaboración una comparativa frente a las implementaciones ya existentes en términos de rendimiento.
- Evaluar el ancho de banda de las transferencias entre un cliente y un servidor del sistema de virtualización de GPUs ejecutándose en plataformas con distintas prestaciones interconectadas mediante redes de alto rendimiento.
- Examinar, desde los puntos de vista del rendimiento y del consumo, la ejecución de varias aplicaciones del ámbito del cálculo científico aceleradas con GPUs sobre distintas configuraciones de clústeres heterogéneos con virtualización de GPUs.

En resumen, la consecución de estos objetivos busca avanzar en una línea de investigación alrededor del desarrollo de rCUDA (*Remote CUDA*), proporcionando nuevos módulos de comunicaciones, con diferentes semánticas, para las redes que soportan el protocolo RDMA, y realizando una evaluación experimental de las posibilidades que ofrecen las actuales tecnologías en el campo de HPC y las alternativas de bajo consumo para la aceleración de aplicaciones científicas utilizando procesadores gráficos remotos. Esto supone un primer paso hacia un objetivo a largo plazo que consiste en el desarrollo de una solución de virtualización para la nueva generación de sistemas de computación de alto rendimiento en la nube.

1.3. Estructura del documento

El resto de la memoria de la tesis se organiza en 4 capítulos. Los párrafos siguientes ofrecen una visión de los contenidos de cada uno de ellos.

El próximo capítulo, el Capítulo 2, describe el estado del arte de las distintas tecnologías en las que se fundamenta el presente trabajo. La primera

sección del capítulo presenta las principales arquitecturas de bajo consumo. Posteriormente se introducen los sistemas de interconexión de alto rendimiento y se continúa con la descripción de la tecnología RDMA. Después, se enumeran las tecnologías de red actuales que soportan el protocolo RDMA. Para terminar, se explora el estado del arte de la virtualización de GPUs y se profundiza en la solución propuesta por rCUDA.

En el Capítulo 3 se aborda el diseño y la implementación de la primera aproximación del módulo de comunicaciones para las tecnologías RDMA, junto con la correspondiente evaluación de su rendimiento. A continuación, se presentan las soluciones completas, y se ofrece una descripción detallada de los mecanismos de conexión y transmisión utilizados en los módulos con semántica de canal y de memoria. Después, se profundiza en los ajustes avanzados de sus parámetros. El capítulo se completa con un análisis comparativo del rendimiento de los nuevos módulos de rCUDA frente a los existentes.

En el Capítulo 4 se ofrece una descripción de las configuraciones CPU-GPU (cliente-servidor) utilizadas para realizar el estudio experimental y que han sido establecidas teniendo en cuenta, tanto la tendencia estándar de uso de HPCCs equipados con procesadores de alto rendimiento, como la aproximación más actual consistente en utilizar procesadores de bajo consumo. Posteriormente, se analiza el rendimiento de las transferencias entre la CPU y la GPU en cada una de las configuraciones. Finalmente se exploran los posibles beneficios y sobrecostes de la virtualización de GPUs mediante una evaluación exhaustiva del comportamiento, en cuanto a consumo y prestaciones, de 4 aplicaciones paralelas aceleradas con GPUs sobre las distintas configuraciones del entorno de experimentación.

Por último, en el Capítulo 5 se ofrecen las conclusiones generales del trabajo desarrollado en esta tesis, se exponen los principales resultados obtenidos, en forma de publicaciones, y se esbozan las líneas abiertas de investigación.

CAPÍTULO 2

Antecedentes

En el presente capítulo se realiza una revisión del estado del arte y de las tecnologías empleadas para llevar a cabo la investigación.

En primer lugar, en la Sección 2.1 se presentan las principales arquitecturas de bajo consumo que se están abriendo paso en la computación de altas prestaciones, o HPC (*High Performance Computing*), como alternativa a los actuales procesadores propósito general Intel Core o Intel Xeon.

Posteriormente, la Sección 2.2 realiza una introducción a los sistemas de interconexión de HPC. Se continua en la Sección 2.3 con la descripción de la tecnología conocida como RDMA (*Remote Direct Memory Access*). Seguidamente, se enumeran sus principales características y las ventajas que aporta frente al resto de mecanismos de transmisión. En el Apartado 2.3.1

se muestra el funcionamiento general de las transferencias de datos con RDMA. A continuación se describen las diferentes semánticas de comunicación y los modos de transporte disponibles para una conexión de RDMA. Se prosigue con la presentación de las interfaces de programación o APIs (*Application Programming Interface*s) que permiten a las aplicaciones tener acceso a las redes con RDMA. En la Sección 2.4, se analizan las tecnologías de red actuales que disponen de capacidad RDMA.

Después, en la Sección 2.5 se describen las alternativas existentes para reducir el número de aceleradores gráficos usados para el cálculo en los clústeres de HPC, o HPCCs (*High Performance Computing Cluster*s). A lo largo de la Sección 2.6 se describe el sistema de virtualización para unidades de procesamiento gráfico, o GPUs (*Graphics Processing Unit*s), denominado rCUDA (*Remote CUDA*). Más en detalle, en los Apartados 2.6.1 y 2.6.3, se presenta la arquitectura cliente-servidor de rCUDA y se explican las técnicas diseñadas para aumentar la eficiencia de las transferencias entre sus componentes. Para terminar, en la Sección 2.7 se presentan las conclusiones de este capítulo.

2.1. Arquitecturas de bajo consumo

El consumo y la eficiencia energética se ha convertido en una de las principales preocupaciones de la comunidad de HPC [5, 27-29]. Con cada generación de computadores se incrementa la potencia computacional, y al mismo tiempo, el consumo energético necesario para hacer funcionar ese sistema [2]. Por lo tanto, es evidente que del mismo modo que es necesario aumentar el rendimiento computacional es imprescindible reducir el consumo energético de esos computadores.

Actualmente, un clúster de alto rendimiento o HPCC está compuesto por procesadores de propósito general, como por ejemplo Intel Core o Intel Xeon. Estos procesadores poseen una gran potencia de cálculo, pero también requieren una gran cantidad de energía eléctrica para su funcionamiento. Algunos estudios recientes han contemplado la utilización de procesadores de bajo consumo como una posible alternativa para aumentar la eficiencia energética de los sistemas de HPC. En particular, los procesadores de las familias Intel Atom y ARM Cortex-A.

En [21] puede encontrarse un extenso estudio sobre la viabilidad de la utilización de procesadores de bajo consumo para HPC. Para ello se evalúan cinco plataformas (Intel Xeon E7, Intel Core i7, Intel Atom, AMD Fusion y ARM Cortex A9) con siete colecciones de tests o benchmarks (Phoronix, CoreMark, Fhourstones, Whetstone, Linpack, OSU y HPL). Sus resultados muestran que el consumo energético del ARM Cortex-A9 es hasta 12 veces menor que el resto de los procesadores.

Por otro lado, en [22] se presenta un análisis comparativo entre un procesador ARM Cortex-A8 y un Intel Atom N330. Sus resultados muestran que el Atom tiene un mejor rendimiento computacional, mientras que el ARM tiene una mayor eficiencia energética. Igualmente, el estudio descrito en [23] realiza una caracterización detallada de varios procesadores de bajo consumo en términos de rendimiento, consumo energético y huella térmica. Los datos demuestran que la plataforma ARM tiene una huella térmica inferior y una mejor eficiencia energética cuando la ejecución utiliza un sólo núcleo. En cambio, en las ejecuciones con varios núcleos, el procesador Intel Atom logra una mejor eficiencia energética.

Por último, destaca la experiencia descrita en [26] sobre la construcción de un HPCC con procesadores ARM. También tiene especial relevancia el proyecto Mont-Blanc [30], cuyo objetivo es el diseño de un nuevo tipo de plataforma HPC construida a partir de soluciones energéticas eficientes como pueden ser dispositivos empotrados y móviles.

A continuación se examinan las dos principales arquitecturas que las investigaciones proponen como alternativas para reducir el consumo de los HPCC.

2.1.1. ARM

ARM es una familia de arquitecturas RISC (*Reduced Instruction Set Computer*) desarrollada por ARM Holdings. Al estar basada en el diseño RISC, los procesadores ARM requieren un menor número de transistores que los procesadores con arquitectura CISC (*Complex Instruction Set Computer*). Esto permite que su coste de fabricación, consumo energético y huella térmica sea menor.

En la actualidad, ARM Holdings desarrolla el conjunto de instrucciones y la arquitectura de los procesadores basados en ARM, pero no fabrica

productos. En su lugar, ARM Holdings vende licencias a terceros para que diseñen productos que implementen las arquitecturas ARM. Entre las principales compañías que fabrican plataformas con arquitectura ARM se encuentran Apple, Broadcom, Nvidia, Qualcomm, Samsung Electronics y Texas Instruments.

Procesadores ARM Cortex-A

Los primeros diseños de procesadores ARM no contaban con una unidad de coma flotante capaz de proporcionar el rendimiento y la latencia requeridas en HPC. Sin embargo, las últimas generaciones de procesadores ARM ya incluyen instrucciones SIMD (*Single Instruction, Multiple Data*), procesadores multinúcleo y una unidad aritmética de coma flotante mejorada. En particular, destacan los siguientes procesadores la familia ARM Cortex-A:

ARM Cortex-A9. Es un procesador superescalar de alto rendimiento que implementa la arquitectura ARMv7. Sus principales características son:

- Implementa la ejecución fuera de orden. Esto le permite ejecutar todas las instrucciones en serie o intentar cambiar el orden para usar todos los bloques funcionales y aumentar su eficiencia.
- Permite configuraciones con múltiples núcleos dentro del mismo encapsulado.
- Dispone de una unidad de coma flotante o FPU (*Floating-Point Unit*) de tipo VFPv3 [31] y una unidad SIMD de coma flotante NEON [32]. La unidad VFPv3 está segmentada y es capaz de ejecutar una sola operación de suma (`ADD`) de doble precisión por ciclo, o una multiplicación (`MUL`) cada dos ciclos. NEON es una unidad SIMD que únicamente soporta operandos enteros y reales de simple precisión. Por lo tanto, un procesador Cortex-A9 a 1 GHz está limitado a un máximo de 1 GFLOPS (10^9 operaciones en coma flotante por segundo).

ARM Cortex-A15. Es un procesador más reciente que el ARM Cortex-A9 y posee una FPU de doble precisión totalmente segmentada [33].

Esto permite que un procesador ARM Cortex-A15 operando a 1 GHz, obtenga un rendimiento de hasta 2 GFLOPS.

ARM Cortex-A57. Implementa el nuevo conjunto de instrucciones ARMv8 [34]. Se caracteriza por utilizar un espacio de direcciones de 64 bits y añade soporte para doble precisión en la unidad NEON. De esta forma, un procesador ARM Cortex-A57 a 1 GHz es capaz de alcanzar 4 GFLOPS. Los resultados presentados en [25] indican que, para algunas aplicaciones de HPC, puede ser una alternativa factible a los procesadores de propósito general tradicionales.

Plataformas Tegra

Tegra es un SoC (*System-on-a-Chip*) desarrollado por NVIDIA para dispositivos portátiles y dispositivos móviles. Los chips Tegra contienen una CPU (*Central Processing Unit*) con arquitectura ARM y una GPU en el mismo encapsulado [35]. La familia Tegra está orientada a realizar tareas de reproducción de audio y vídeo con un bajo consumo energético. Los modelos más recientes son:

Tegra 3. Combina la CPU ARM con una GPU de bajo consumo "Fermi" para aumentar el rendimiento. El procesador integrado en el SoC Tegra 3 es un ARM Cortex-A9 con cuatro núcleos que utiliza la tecnología de vSMP (*Variable Symmetric Multiprocessing*) para añadir un quinto núcleo ARM (conocido como "núcleo acompañante") que funciona a una frecuencia más baja y con un consumo energético menor [36]. En función de los requisitos de las aplicaciones es posible desconectar los cuatro núcleos principales y que la plataforma funcione únicamente con el núcleo acompañante, que consume menos energía. Con el fin de hacer transparente este proceso se implementan mecanismos de gestión en el propio hardware y en los controladores del sistema operativo.

Tegra K1. Es la última evolución de Tegra 3 disponible en el mercado. Tegra K1 incorpora una CPU ARM Cortex-A15 con vSMP y una GPU con arquitectura "Kepler" [37].

2.1.2. Intel Atom

Atom es la familia de procesadores de bajo consumo fabricada por Intel. Está orientada a portátiles de coste reducido o servidores de bajo consumo. Sus frecuencias van desde 800 MHz a 2,66 GHz.

Excepto los primeros modelos del Intel Atom (versiones N2xx y Z5xx), este tipo de arquitecturas implementa el conjunto de instrucciones x86-64 y x86 (IA-32). El hecho de tener una arquitectura basada en x86 los han convertido en uno de los candidatos más adecuados para sustituir a los procesadores de gran consumo Intel Core e Intel Xeon, en ciertas aplicaciones.

Los procesadores Intel Atom están basados en la microarquitectura Bonnell, que puede ejecutar hasta dos instrucciones por ciclo. Cada instrucción x86 (CISC) se traduce en microinstrucciones internas más simples (RISC) antes de su ejecución. Las microinstrucciones internas pueden contener tanto una carga como un almacenamiento en memoria relacionados con una operación de la unidad aritmético-lógica o ALU (*Arithmetic Logic Unit*). Esto permite al procesador realizar varias tareas por ciclo de reloj.

El procesador dispone de una ejecución segmentada en 16 etapas, donde cada etapa se descompone en tres partes: decodificación, ejecución y acceso a la caché.

El procesador Atom dispone de dos ALUs y dos FPUs. La primera ALU se utiliza para ejecutar cualquier operación de desplazamiento de bits mientras que la segunda se encarga de los saltos. Las FPUs se utilizan para cualquier operación aritmética, incluyendo las relacionadas con enteros. La primera FPU se utiliza únicamente para las operaciones de adición, mientras que la segunda FPU ejecuta las operaciones de multiplicación, división y SIMD. Generalmente, las operaciones básicas se pueden completar en un solo ciclo de reloj, mientras que el procesador puede consumir hasta 31 ciclos en instrucciones más complejas, como por ejemplo las divisiones de coma flotante. Esto permite, por ejemplo, que un procesador Atom N270 a 1,60 GHz con un sólo núcleo obtenga un rendimiento de 2,1 GFLOPS [20].

Para terminar, cabe señalar que los últimos modelos destinados a servidores de bajo consumo disponen de 8 núcleos compatibles con la tecnología Hyper-Threading [38].

2.2. Redes de interconexión de altas prestaciones

Hoy en día, un HPCC está constituido básicamente por varios ordenadores independientes que están interconectados por una infraestructura de red y que se gestionan de forma centralizada por un software especializado.

El sistema de interconexión que forma parte del HPCC permite compartir la información entre los nodos y es usado como vehículo para el seguimiento y control del clúster. Evidentemente, desde el punto de vista de HPC, el tráfico de mayor importancia es el relacionado con la computación, es decir, los datos de una aplicación que comparten los nodos. Sin embargo, la red también debe acomodar los tráficos de almacenamiento, gestión y planificación. Teniendo en cuenta esto, la mayoría de los HPCC disponen de al menos dos redes dedicadas [18, 39]:

- Una red de gestión, planificación y almacenamiento, y
- Una red de cómputo.

La única tarea de la red de cómputo es transportar el tráfico de cálculo. Por ello tiene un impacto directo en el rendimiento computacional del clúster y la convierte en uno de los elementos más importantes del diseño de los HPCCs.

En función de la cantidad de datos intercambiados, las redes 1GbE (Ethernet de 1 Gbit/s) basadas en TCP/IP pueden convertirse en la solución para transportar el tráfico computacional. Sin embargo, debido a la sobrecarga de este protocolo y su alta latencia, pueden no ser la mejor elección para la mayoría de aplicaciones de HPC.

Para las aplicaciones que requieren una latencia baja y un elevado ancho de banda se dispone de redes tales como IB (InfiniBand), 10GbE (Ethernet de 10 Gbit/s), Myricom Myrinet, Quadrics QsNet y Dolphin SCI [18]. Estos sistemas de interconexión de altas prestaciones proporcionan mecanismos más eficientes para enviar mensajes que 1GbE.

2.3. RDMA

Entre las redes de altas prestaciones más utilizadas para un HPCC que se han comentado en la Sección 2.2 se encuentran varias tecnologías que

soportan RDMA. Éste es un método relativamente nuevo para interconectar plataformas, capaz de proporcionar un rendimiento superior al que se obtiene con redes tradicionales tales como TCP/IP sobre Ethernet.

Para acceder a una tecnología RDMA es necesario que el equipo disponga de un adaptador de canal, o CA (*Channel Adapter*), conectado al bus PCIe (*PCI-Express*). Un CA no sólo es el punto de unión de un nodo con la infraestructura de red RDMA sino que además implementa en hardware toda la lógica necesaria para ejecutar el protocolo de RDMA sobre un medio de conexión.

Cuando una aplicación emplea una tecnología de red RDMA obtiene los siguientes beneficios frente a las redes tradicionales basadas en TCP/IP (véase la Figura 2.1):

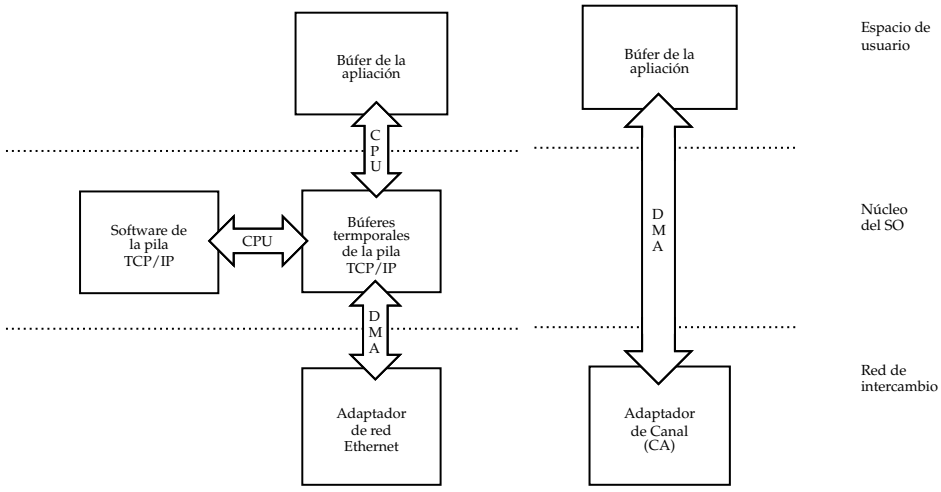
Eliminación de copias. Las transferencias de datos se realizan sin utilizar el software de la pila de red del SO (Sistema Operativo). De esta forma, los datos se envían directamente a los búferes de destino, eliminando la necesidad de copiar datos entre la memoria de la aplicación y el SO [39].

Transmisión sin pasar por núcleo del SO. En las redes tradicionales las aplicaciones realizan peticiones al núcleo del SO a través de una API de red. Sin embargo, las API de RDMA permiten a los procesos de usuario solicitar los servicios directamente al CA, sin tener que ejecutar llamadas al núcleo del SO. Esto reduce el número de cambios de contexto entre el espacio del núcleo y espacio de usuario [40].

Soporte de operaciones de la semántica de acceso a memoria. Junto con las operaciones de envío y recepción, las aplicaciones pueden emplear operaciones RDMA de escritura y lectura sobre la memoria remota.

Agrupación de búferes en una sola transferencia. RDMA dispone de operaciones de transferencia de datos que permiten el envío de múltiples búferes de memoria como un único mensaje o la distribución del conjunto de datos recibidos entre múltiples búferes de memoria [41].

Conservación de los límites del mensaje transmitidos. A diferencia de lo que ocurre en las redes orientadas a la transmisión de flujos de datos (como por ejemplo las redes basadas en TCP/IP), RDMA permite



(a) TCP/IP **(b)** RDMA

Figura 2.1: Comparación de RDMA con las redes tradicionales basadas en TCP/IP.

que los mensajes sean tratados como unidades discretas y puedan ser transmitidos por las aplicaciones sin necesidad de dividir su envío en varias transacciones.

Ejecución sin la intervención de la CPU. Las operaciones RDMA permiten a una aplicación leer o escribir en la memoria de un equipo remoto. La aplicación remota únicamente debe registrar el búfer de memoria destino al inicio de la ejecución. De este modo las operaciones de comunicación se ejecutan sin la intervención de la CPU remota.

En resumen, RDMA proporciona a las aplicaciones un mecanismo para realizar transferencias de datos con una baja latencia y un elevado ancho de banda sin añadir sobrecarga a la CPU [42].

2.3.1. Modelo funcional

En primer lugar, una aplicación que utiliza las operaciones de transferencia de RDMA debe reservar una zona de la memoria principal con la paginación bloqueada ("memoria no paginable") [43]. De esta forma indica al núcleo del SO que la memoria pertenece a la aplicación, y que no tiene que moverse al espacio de intercambio (*swapping*).

A continuación, la aplicación registra la zona de memoria para crear la región de memoria registrada o MR (*Memory Region*) que utilizará en

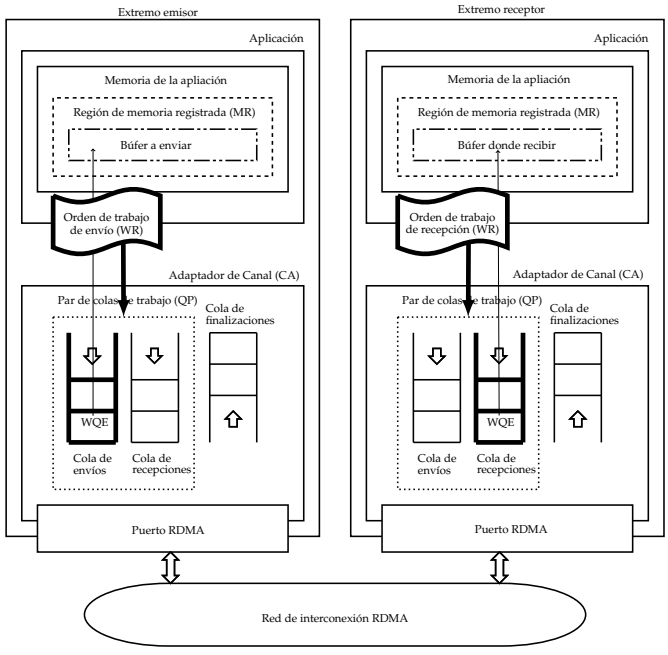


Figura 2.2: Publicación de una solicitud de transferencia RDMA.

las operaciones RDMA de transferencia. El proceso de registro consiste en informar al CA de la dirección de la zona de memoria no paginable, y en preparar el transporte de datos entre esa zona y el dispositivo RDMA utilizando la técnica de acceso directo a memoria o DMA (*Direct Memory Access*).

Posteriormente se crean las colas que se emplearán en las comunicaciones RDMA: la cola de envío, la cola de recepción y la cola de finalización.

Las dos primeras son las responsables de planificar y realizar las transferencias, por lo que reciben el nombre de colas de trabajo. Ambas estructuras se crean conjuntamente formando un par que en la nomenclatura RDMA se denomina QP (*Queue Pair*). Cada QP debe ser inicializado en los dos extremos de la conexión.

La cola de finalización o CQ (*Completion Queue*) se utiliza para almacenar las notificaciones de las transferencias que han sido completadas, por lo que se encuentra asociada al menos a una de las colas de trabajo.

Una vez las tres colas están operativas, como se observa en la Figura 2.2, la aplicación genera una solicitud de trabajo o WR (*Work Request*) para indicar al CA que debe realizar una transferencia. Cada WR crea

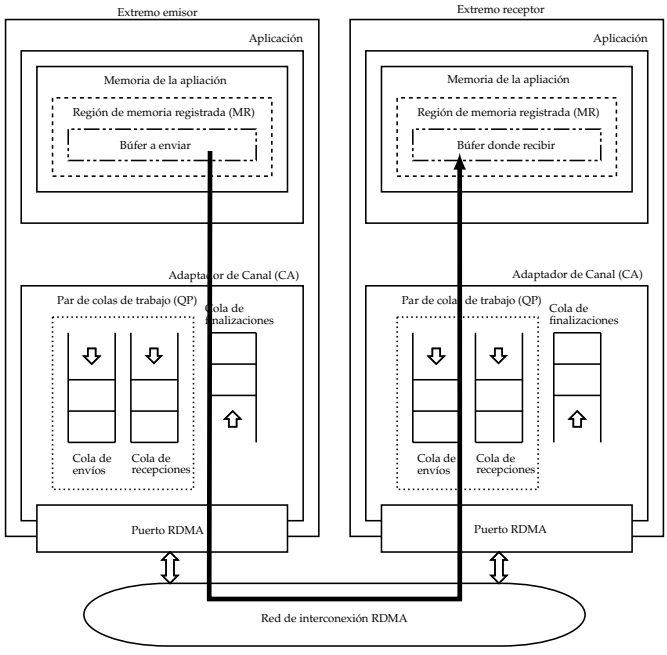


Figura 2.3: Transmisión de los datos usando el modelo de RDMA basado en colas.

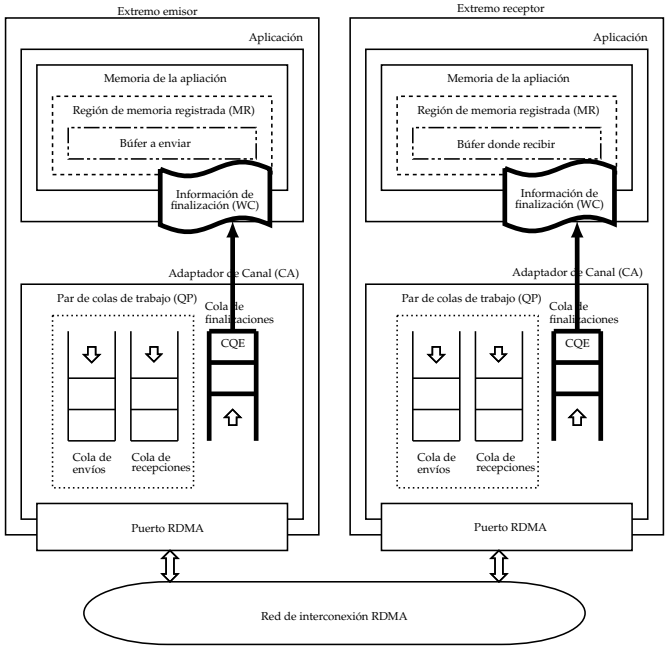


Figura 2.4: Finalización de una solicitud de transferencia RDMA.

un WQE (*Work Queue Element*) en la cola correspondiente del par de colas. Un WQE es una estructura de datos que, entre otros valores, contiene la dirección de la MR que se va a enviar o recibir.

Después, el adaptador procesa las WQE en orden y realiza las transferencias de los búferes indicados (véase la Figura 2.3). Al finalizar una transacción se elimina el WQE asociado, y como muestra la Figura 2.4, se genera un nuevo CQE (*Completion Queue Element*) en la CQ que recoge el estado de finalización.

La aplicación sondea periódicamente el estado de la CQ para detectar cuándo se ha completado una WR. Si existe un nuevo CQE, lo desencola y examina su contenido. De esta forma obtiene el estado de finalización o WC (*Work Completion*) asociado a la WR que se ha ejecutado.

2.3.2. Verbos

La especificación del protocolo RDMA se realiza en función de "verbos" o *verbs*. Los verbos ofrecen una definición abstracta de la funcionalidad que un adaptador suministra a un equipo, y son enviados por las aplicaciones al QP en forma de WR. Aunque existen varios tipos de verbos, los más relevantes para este trabajo son los relacionados con la lectura remota (*READ*), la escritura remota (*WRITE*), el envío (*SEND*) y la recepción (*RECV*).

Generalmente los verbos se añaden a la cola de envío del QP (excepto *RECV*, que se inserta en la cola de recepción). Para enviar un verbo al CA, una aplicación realiza una llamada al controlador del adaptador sin salir del espacio de ejecución del usuario. Entonces, el controlador genera un WQE y lo publica en el cola correspondiente del QP. Cuando el CA completa las tareas de red asociados con el verbo, crea un CQE en la CQ asociada al QP.

Debido a que los verbos especifican el comportamiento del CA, éstos pueden influir en el diseño del software que proporciona el acceso al CA, como por ejemplo las APIs. Sin embargo, la especificación de RDMA no define explícitamente ninguna API. Los verbos sólo realizan una descripción semántica del comportamiento de las operaciones sin ofrecer detalles operativos ni sintácticos. Corresponde a los fabricantes y desarrolladores de software fijar su propia API usando las semánticas proporcionadas por los verbos.

En definitiva, los verbos ofrecen la definición semántica de la API que facilita a las aplicaciones el acceso directo al CA.

2.3.3. Semánticas de comunicación

Dentro de la especificación de RDMA pueden encontrarse dos tipos de semánticas de verbos para las transferencias de datos:

Semántica de memoria. Es la semántica de los verbos *WRITE* y *READ*. Ambos verbos especifican una dirección de la memoria del extremo remoto para operar sobre ella. Esto provoca que las transferencias se vean como simples accesos a memoria y no como transmisiones de mensajes. En la semántica de memoria se dice que el extremo destino es pasivo, ya que la CPU del receptor no es consciente de la operación de transferencia [41, 44].

Semántica de canal. Es propia de los verbos *SEND* y *RECV* por lo que también se conoce como "semántica de envío y recepción". Cuando se utiliza el verbo *SEND*, el mensaje se escribe en un búfer indicado por el receptor mediante un *RECV* que previamente ha publicado. Como resultado, el emisor no tiene visibilidad de los búferes ni de las estructuras de datos que se encuentran en lado de destino. Así pues, el emisor sólo puede enviar un mensaje si el destinatario ha publicado la recepción. Por tanto, a diferencia de *WRITE* y *READ*, la CPU del extremo receptor sí participa en la transferencia.

Por último, se debe señalar que, aunque los verbos *SEND* y *RECV* son técnicamente verbos de RDMA, normalmente se emplea el término "verbos de mensajería" para referirse a ellos; por otra parte, el término "verbos RDMA" se utiliza para designar a *WRITE* y *READ*.

2.3.4. Modos de transporte

En la especificación de RDMA se fijan cuatro modos de transporte que pueden ser usados en las comunicaciones:

Conexión fiable o RC (*Reliable Connection*). En este modo cada uno de los extremos crea un QP dedicado a esa comunicación y los mensajes son entregados de forma fiable y en orden desde la cola de envío de

un QP a la cola de recepción del otro QP. Una conexión RC es similar a una conexión TCP.

Conexión no fiable o UC (*Unreliable Connection*). Realiza el envío de forma no fiable por lo que pueden existir pérdidas de mensajes, de modo que es responsabilidad de un protocolo de nivel superior realizar un control de errores. No obstante, es necesario que los extremos dispongan de un QP dedicado a realizar las transferencias entre ambos.

Datagrama no fiable o UD (*Unreliable Datagram*). Permite que un QP transmita un mensaje a cualquier otro QP de tipo UD por lo que no requiere la creación de QPs dedicados. Una conexión UD se puede comparar con una conexión de UDP (*User Datagram Protocol*) ya que en este modo no se garantiza ni el orden ni la recepción de los datos.

Datagrama fiable o RD (*Reliable Datagram*). Este modo es similar a RC pero no precisa QP dedicados. Actualmente, RD no está soportado por el hardware de ninguna de las tecnologías de interconexión existentes.

En resumen, los modos de transporte basados en conexión realizan una asociación unívoca entre dos equipos. En cambio, en los dos modos de tipo datagrama, un equipo tiene libertad para comunicarse con cualquier otro dispositivo de la red; es decir, no existe una vinculación que una los extremos a un canal concreto de comunicación.

Por otra parte, se etiqueta un modo como fiable cuando realiza la transmisión de los mensajes de forma confiable y ordenada. El escenario contrario ocurre en los modos no confiables que pueden perder paquetes, y donde es responsabilidad de los protocolos superiores realizar la gestión de errores.

2.3.5. Interfaz de programación de aplicaciones

Para que las aplicaciones puedan acceder a los servicios RDMA, la descripción semántica del comportamiento proporcionada por los verbos debe concretarse en una API que implemente ese comportamiento. Sin embargo, como se ha indicado en el Apartado 2.3.2, la especificación de RDMA no describe directamente ninguna API, sólo define las conductas

requeridas a la API. De esta forma se permite a las organizaciones desarrollar una API para RDMA asegurando la interoperabilidad entre diferentes proveedores y plataformas.

Un buen ejemplo es la OFA (*OpenFabrics Alliance*), que junto a los proveedores comerciales, ha desarrollado la definición sintáctica de unas APIs para RDMA que se integran dentro del proyecto OFED (*OpenFabrics Enterprise Distribution*) [45]. Como su nombre indica, OFED es un software de código abierto, que se encuentra a disposición de la comunidad bajo las licencias BSD y GNU. El proyecto OFED recibe el respaldo de los fabricantes de hardware RDMA, de los proveedores de SO y de la comunidad de Open Source de Linux. En consecuencia, las APIs de OFED han sido ampliamente aceptadas y existe gran cantidad de información disponible en Internet (como por ejemplo [46, 47]).

Del conjunto de APIs que incorpora OFED para acceder a los servicios de RDMA destacan las siguientes:

`ibverbs`. Ofrece a las aplicaciones la funcionalidad necesaria para que puedan interactuar directamente con el controlador del CA sin salir del espacio de ejecución de usuario. Cada verbo de la especificación de RDMA dispone de una función equivalente dentro de la API que implementa la semántica de comportamiento del verbo [43]. Sin embargo, como se observa en la Tabla 2.1, la API fija limitaciones para algunas operaciones en función del modo de transporte escogido para la comunicación.

		UD	**UC**	**RC**
Operación	Envío	✓	✓	✓
	Recepción	✓	✓	✓
	Escritura	✓	✓	✓
	Lectura		✓	✓
Tamaño máximo del mensaje		MTU	2GB	2GB

Tabla 2.1: Limitación de las operaciones de la API ibverbs.

`rdmacm`. Proporciona una interfaz de transporte RDMA para establecer conexiones [43]. La API se basa en conceptos propios de los sockets que han sido adaptados a las semánticas del QP, como por ejemplo

las funciones de establecimiento de conexión empleadas en el modelo cliente-servidor. Trabaja en conjunto con la API `ibverbs`.

Para utilizar las operaciones de transferencia de datos de la API `ibverbs`, las aplicaciones deben completar en orden estos pasos:

1. Crear el QP, la CQ y las estructuras de datos de la API.
2. Obtener la dirección del QP remoto.
3. Comunicar al otro extremo la dirección del QP local.
4. Preparar el QP para la transmisión RDMA realizando cambios en su estado.

El cuarto paso es particularmente complejo y tedioso para el desarrollador de aplicaciones. Esto se debe en parte al confuso proceso de transición de los estados del QP y al gran número de parámetros que requieren las llamadas de la API. Para superar este problema, la API `rdmacm` suministra una abstracción de alto nivel basada en dirección IP y número de puerto, que es similar a la empleada por las APIs de los sockets TCP/IP. Al usar la API `rdmacm` se puede hablar pues de un extremo pasivo (servidor) y un extremo activo (cliente) de la conexión.

2.4. Tecnologías RDMA

Actualmente, como muestra la Figura 2.5, existen tres tecnologías que soportan los verbos de la especificación de RDMA: InfiniBand, iWARP y RoCE. En esta sección se ofrece una descripción de cada una de ellas y de sus principales características.

2.4.1. InfiniBand

IB (InfiniBand) es una tecnología de interconexión de alta velocidad, baja latencia y reducida sobrecarga para la CPU, que surge a finales de los años noventa de la fusión de dos iniciativas existentes: Future I/O, desarrollada por Compaq, IBM y Hewlett-Packard; y de Next Generation E/S, creada por Intel, Microsoft y Sun Microsystems. Como resultado se funda en 1999 la IBTA (*InfiniBand Trade Association*), que incluye entre otros

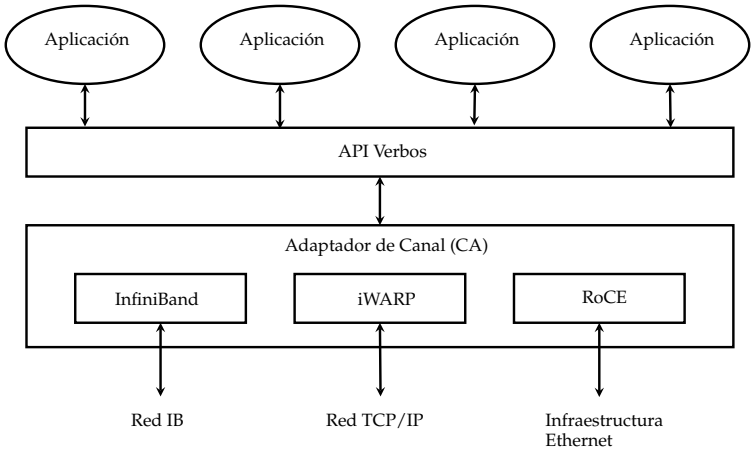


Figura 2.5: Tecnologías de red RDMA.

a Compaq, Dell, HP, IBM, Intel, Microsoft y Sun, para que desarrolle y mantenga las especificaciones de IB [48].

En un principio, el diseño de IB se realizó con el objetivo de ser una red unificada capaz de superar las limitaciones de los buses locales existentes y de estandarizar tecnologías propietarias en los HPCC, como por ejemplo, Servernet, Myricom, Giganet . . . Hoy en día, a pesar de no haberse convertido en esa tecnología de red multiuso, IB es la solución más utilizada en los grandes HPCC. En concreto, como se observa en la Figura 2.6, supone el 44,8 % de los sistemas de interconexión de la lista TOP500 [18].

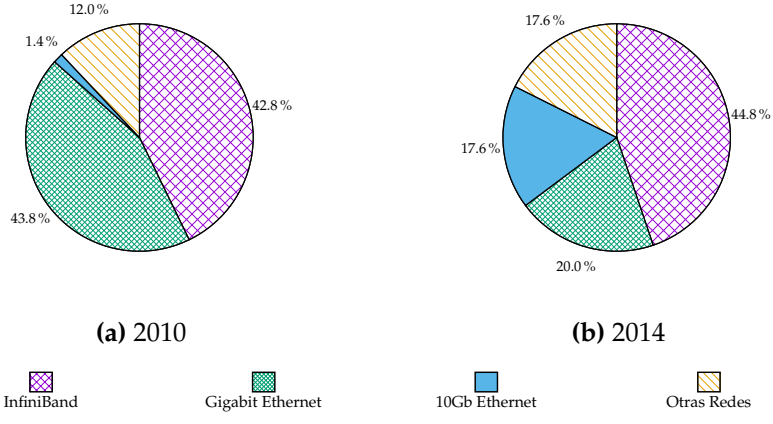


Figura 2.6: Evolución de las infraestructuras de red en la lista TOP500.

La especificación de IB define una topología conmutada, donde los nodos se conectan a la red a través de un adaptador que recibe el nombre de HCA (*Host Channel Adapter*). Estos adaptadores transmiten los datos en paquetes de hasta 4 KB que se agrupan para formar mensajes [49]. Un mensaje puede consistir en una operación de lectura o escritura remota, una operación de envío, una operación atómica o una transmisión multicast.

IB ofrece las siguientes ventajas frente a otras soluciones de red:

- Soporta RDMA de forma nativa.
- Puede alcanzar velocidades de hasta 100 Gbit/s por puerto en su última especificación, conocida como EDR (*Enhanced Data Rate*) [50].
- Proporciona una latencia en las comunicaciones entre aplicaciones por debajo de 1,2 µs [51].

Sin embargo, IB tiene el inconveniente de que si se compara con otras tecnologías de interconexión, como por ejemplo Ethernet, posee un coste económico extremadamente elevado [52].

2.4.2. Tecnologías RDMA sobre Ethernet

Desde su implantación todas las instalaciones informáticas de cálculo disponen de infraestructuras Ethernet, pero para aumentar su rendimiento se suele incorporar una infraestructura de red de alta velocidad dedicada al transporte de los datos de cómputo, como por ejemplo IB. Esto implica que los administradores de red deben gestionar dos infraestructuras, lo que aumenta el tiempo y el esfuerzo dedicado al mantenimiento de la red. Además, al añadir el nuevo hardware y mantener el existente para Ethernet hace que el consumo energético se incremente [53].

En los últimos años, con 10GbE, la red Ethernet ha conseguido reducir la latencia en los adaptadores y ha mejorado su ancho de banda. Al amparo de estas circunstancias han surgido las soluciones iWARP (*Internet Wide Area RDMA Protocol*) [54] y RoCE (*RDMA over Converged Ethernet*) [55], que ofrecen RDMA sobre 10GbE. Esto ha hecho posible que Ethernet se posicione como una alternativa de menor complejidad y coste que IB para los HPCC [56, 57]. En particular, como puede verse en la Figura 2.6, 10GbE se ha convertido en la tercera tecnología de interconexión para los HPCC de la lista TOP500 [18].

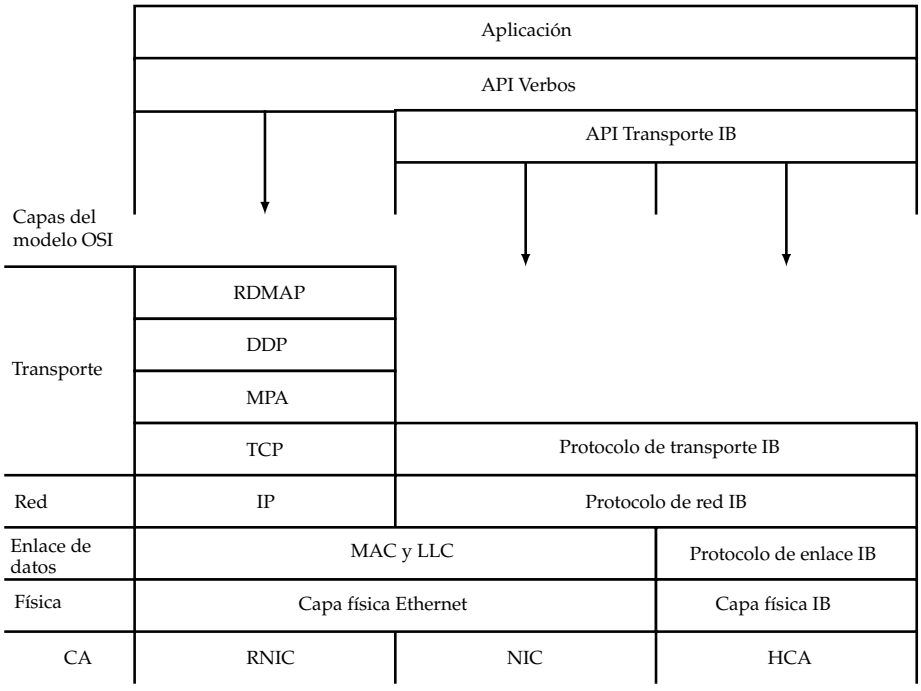


Figura 2.7: Pila de protocolos de las tecnologías RDMA.

iWARP

El estándar iWARP (*Internet Wide Area RDMA Protocol*) fue presentado en 2002 y desde entonces ha tenido una expansión reducida. iWARP utiliza el protocolo DDP (*Direct Data Placement Protocol*) definido por IETF (*The Internet Engineering Task Force*) para proporcionar servicios RDMA sobre una red TCP/IP estándar, de forma que los servicios DDP se encapsulan sobre la capa de transporte existente. Sin embargo, para poder usar DDP sobre TCP, se requiere una técnica conocida como MPA (*Marker PDU Aligned*), que está orientada a garantizar los límites de los mensajes. DDP no está destinado a ser utilizado directamente por la API de RDMA por lo que debe añadirse un nuevo protocolo por encima. Este protocolo se llama RDMAP (*RDMA Protocol*) y proporciona los servicios para leer y escribir datos remotos.

Por tanto, como se puede observar en la Figura 2.7, la especificación de RDMA sobre TCP de iWARP es en realidad RDMAP sobre DDP, operando este último sobre MPA y TCP [58]. Todos estos protocolos se implementan

en el hardware del adaptador de iWARP instalado en el equipo, que recibe el nombre de RNIC (*RDMA Network Interface Controller*) [59]. No obstante, a cambio de renunciar a los beneficios que proporciona RDMA, es posible utilizar iWARP con adaptadores Ethernet convencionales si se ejecuta el software SoftRDMA [60, 61] en el *host*.

La utilización de iWARP ofrece ciertas ventajas frente a otras tecnologías de interconexión:

- Los resultados de la pruebas sobre iWARP demuestran que posee una latencia de 6 µs [62].
- Se crea una red unificada donde el tráfico de LAN (*Local Area Network*), SAN (*Storage Area Network*) y RDMA se transmite sobre un único medio. Además, las comunicaciones de las aplicaciones y el tráfico de gestión convergen reduciendo el número de cables, dispositivos de conexión y conmutadores.
- Los administradores pueden utilizar las herramientas habituales de una red TCP/IP para gestionar el tráfico de la red iWARP.
- Como iWARP utiliza Ethernet y funciona sobre la pila de protocolos estándar de TCP/IP, es capaz de operar con el equipamiento estándar. De manera que los mensajes pueden ser enrutados a través de las subredes TCP/IP usando las infraestructuras existentes [63].

A pesar de esto, iWARP también presenta una serie de problemas:

- El tráfico DDP no puede ser fácilmente administrado y optimizado por la infraestructura, lo que genera problemas de eficiencia. Esto es debido a que no se proporciona a la capa de transporte ningún mecanismo para identificar el tráfico RDMA. Como resultado, los adaptadores de red no pueden diferenciar si un mensaje transporta RDMA o TCP por si solos.
- La implementación hardware es complicada a causa de que DDP ha sido diseñado para funcionar en la capa de transporte TCP/IP de una red Ethernet. La capa de enlace de datos de Ethernet ofrece un servicio *best effort* y es posible que se pierdan paquetes, por lo que se

depende de la capa de transporte para la obtención servicios confiables. Así pues es necesario que el hardware del RNIC proporcione las herramientas para realizar esa tarea.

- iWARP es una tecnología antigua y actualmente muchos adaptadores modernos con otra tecnología pueden conseguir una latencia similar.

RoCE

Los recientes mejoras que ha introducido DCB (*IEEE Data Center Bridging*) en la capa de enlace de datos de Ethernet han posibilitado la aplicación de servicios avanzados de transporte RDMA directamente sobre Ethernet. DCB proporciona mecanismos de control de congestión y de flujo por lo que algunas de las funciones ofrecidas por TCP y DDP son redundantes. De esta forma, muchos de los servicios ofrecidos en la pila iWARP son innecesarios si se aplica DCB. Como resultado, se planteó la cuestión de rediseñar iWARP para conseguir una aproximación que funcionara de forma más eficiente sobre DCB. Así es como surge la tecnología conocida como RoCE (*RDMA over Converged Ethernet*), que fue estandarizada por el IBTA en 2010.

Entre las ventajas que ofrece RoCE destacan las siguientes:

- RoCE utiliza los avances que proporciona Ethernet (en particular DCB) y, a diferencia de iWARP, funciona directamente sobre la capa de transporte.
- Los conmutadores de red pueden trabajar con RoCE de forma nativa.
- Con el hardware adecuado es posible alcanzar hasta 40 Gbit/s [64].
- RoCE tiene una latencia teórica de 1,6 µs [55].
- Al igual que iWARP se crea una red unificada donde se reducen el número de cables, dispositivos de conexión y conmutadores.
- Cualquier equipo con adaptadores no RoCE puede conectarse a la infraestructura usando el software SoftRoCE [65]. Sin embargo estos dispositivos no pueden aprovechar las ventajas que ofrece la tecnología RDMA.

No obstante, RoCE presenta los siguientes inconvenientes:

- El tráfico RoCE no puede ser enrutado por una red TCP/IP antigua.
- Para poder utilizar RoCE en una infraestructura es necesario un conmutador DCB con soporte PFC (*Priority Flow Control*).

2.5. Virtualización de GPUs para computación general

Actualmente, los HPCCs están equipados con redes de gran ancho de banda y baja latencia, múltiples nodos multinúcleo, sistemas de almacenamiento distribuido, etc. [66]. Además, aquellos HPCCs que buscan obtener mayor rendimiento disponen de aceleradores hardware, como puede verse en la lista TOP500 [18]. Esto les permite ejecutar de forma más rápida ciertos cálculos que tradicionalmente se llevaban a cabo en las CPUs [67]. Cuando los aceleradores hardware para el cálculo son las GPUs, se habla de GPGPU (*General-Purpose computing on Graphics Processing Units*). El motivo de esta elección es que los aceleradores gráficos disponen de gran potencia de cálculo y un alto grado de paralelismo. Sin embargo, debido a su elevado coste y consumo [68], es habitual que no todos los nodos del HPCC se encuentren físicamente equipados con una GPU. Para poder superar esta limitación, los HPCCs disponen de dos alternativas: compartir las GPUs o virtualizar las GPUs.

Dentro del primer caso, se pueden encontrar soluciones comerciales que permiten conectar varios nodos del HPCC a un chasis externo que agrupa varias GPUs, como por ejemplo la serie de productos NextIO de la compañía vCORE [69]. Al compartir varias GPUs es posible ofrecer una configuración donde existen más nodos con acceso a un acelerador que GPUs están presentes en el HPCC. No obstante, esta opción se encuentra limitada por su elevado coste de adquisición y el reducido número de conexiones físicas del chasis.

Por el contrario, el uso de la virtualización de GPUs para GPGPU permite superar estos contratiempos, y al mismo tiempo reduce los costes energéticos, de mantenimiento, de administración y de adquisición [70, 71]. Los sistemas que implementan esta solución se centran en virtualizar una

de las dos APIs de alto nivel disponibles para programar los aceleradores gráficos:

OpenCL (*Open Computing Language*). Es una alternativa de software libre para la programación paralela [17]. La principal ventaja de OpenCL es que permite escribir programas que se pueden ejecutar en un conjunto heterogéneo de plataformas que incluye a las GPUs. Entre los proyectos de virtualización de la API de OpenCL se encuentran:

- dOpenCL [72] es una solución que integra de forma transparente los nodos de un sistema distribuido con aceleradores en una sola plataforma OpenCL. El sistema de ejecución dOpenCL está diseñado como un software distribuido que comprende el controlador dOpenCL del cliente, el demonio dOpenCL y la biblioteca de comunicaciones dOpenCL. El controlador del cliente está instalado en el ordenador origen, y el demonio se ejecuta en cada nodo de cómputo del sistema distribuido objetivo. La biblioteca de comunicación está instalada en el equipo origen y en los nodos de cómputo, y facilita la comunicación entre el controlador del cliente y los demonios.
- VCL [73] es un envoltorio o *wrapper* para OpenCL que permite a las aplicaciones utilizar de forma transparente múltiples dispositivos OpenCL remotos como si fueran locales.
- SnuCL [74] extiende la semántica original de OpenCL con el entorno de clúster heterogéneo con aceleradores.
- VOCL [75] está basado en el modelo de programación OpenCL y expone las GPUs físicas como recursos virtuales deslocalizados que se pueden gestionar de forma transparente.

CUDA (*Compute Unified Device Architecture*). Es la arquitectura de cálculo paralelo de NVIDIA. CUDA dispone de una plataforma software que permite programar las GPUs [76]. Para ello, proporciona una API [16] que ofrece un conjunto de extensiones para el lenguaje de programación C [77-79].

Hoy en día, gran parte de las aplicaciones escogen CUDA para GPGPU [80, 81]. Esto es debido esencialmente a que OpenCL no ofrece

acceso directo a ciertas ventajas hardware que disponen las GPUs (como por ejemplo el acceso a la memoria no paginable o *pinned memory*). Lo que supone una restricción importante si se tiene en cuenta que las GPUs utilizan estos mecanismos para acelerar el cálculo.

Existe un notable número de aproximaciones para la virtualización de GPUs a través de la API de CUDA:

- vCUDA [82] implementa una pequeña parte de las funciones de la versión 1.1 de la API de CUDA (la última versión a fecha de junio de 2015 es la 6.5). Asimismo, vCUDA emplea XML-RPC en las comunicaciones, lo que repercute de forma negativa en el rendimiento de la solución, ya que es necesario realizar etapas de codificación y decodificación durante la transmisión.

- VGPU [83] es un software comercial del que no existe información detallada. Además, no es posible obtener una versión pública que pueda ser utilizada con el propósito de realizar pruebas.

- gViM [84, 85] está específicamente diseñado para ofrecer GPUs a máquinas virtuales basadas en Xen.

- DS-CUDA [86] es una herramienta de virtualización remota de GPUs. Soporta la API de CUDA 4.1.

- rCUDA [19, 87, 88] es actualmente la solución de virtualización de aceleradores gráficos más robusta, flexible y eficiente. Destaca notablemente sobre el resto porque; a diferencia de vCUDA, gVIM y DS-CUDA; rCUDA implementa operaciones asíncronas y soporta la funcionalidad de CUDA 6.5 (excepto las funciones gráficas) [89]. Esto permite que cualquier aplicación CUDA pueda utilizar los servicios de los aceleradores gráficos ubicados en otras máquinas sin necesidad de modificaciones. Además, a diferencia de VGPU y gViM, rCUDA es capaz de proporcionar acceso a estos dispositivos tanto desde máquinas físicas como virtuales [90].

2.6. rCUDA

En los siguientes apartados de esta sección se describen las características y funcionalidades de rCUDA, así como ciertos detalles de su diseño e implementación, relevantes a efectos de esta tesis.

2.6.1. Arquitectura

Como se observa en la Figura 2.8, la solución rCUDA es un software que utiliza el modelo de aplicación distribuida cliente-servidor con el fin de proporcionar acceso a GPUs remotas para realizar GPGPU en SO Linux de 32 o 64 bits [19, 88]. En esencia, como muestra la Figura 2.9, rCUDA ofrece acceso en el nodo local (cliente) a las GPUs remotas mediante una biblioteca que reemplaza a la API de CUDA. Esta biblioteca intercepta las llamadas de la API y las redirige a un nodo remoto (servidor) equipado con una o varias GPUs donde son ejecutadas [87, 91]. Así, en función de la infraestructura existente en el HPCC, se puede desplegar el servidor de rCUDA sobre cualquier nodo que disponga de al menos una GPU, o sobre un nodo dedicado y especializado con varias GPUs [70, 71].

Cliente

Los nodos que requieren los servicios de una GPU deben instalar el software de cliente de rCUDA. Este software contiene una biblioteca que intercepta, procesa y reenvía al servidor rCUDA las llamadas a la API de CUDA que se realizan desde las aplicaciones [92]. Al comienzo de la ejecución de una aplicación, cuando el enlazador dinámico del SO carga la biblioteca, se establece una conexión con el servidor. Durante la ejecución, cada una de las llamadas a la API de CUDA crea una petición que se envía al servidor. En el caso de que la petición corresponda a una función de CUDA síncrona, la biblioteca bloquea la ejecución después de realizar envío, y espera la respuesta del servidor. Al finalizar la aplicación, la biblioteca cierra la conexión y libera los recursos reservados.

Para poder enviar peticiones a través de varios medios de transmisión, el software de rCUDA dispone de dos módulos de comunicaciones, el módulo TCP y el módulo IB, que proporcionan la funcionalidad para

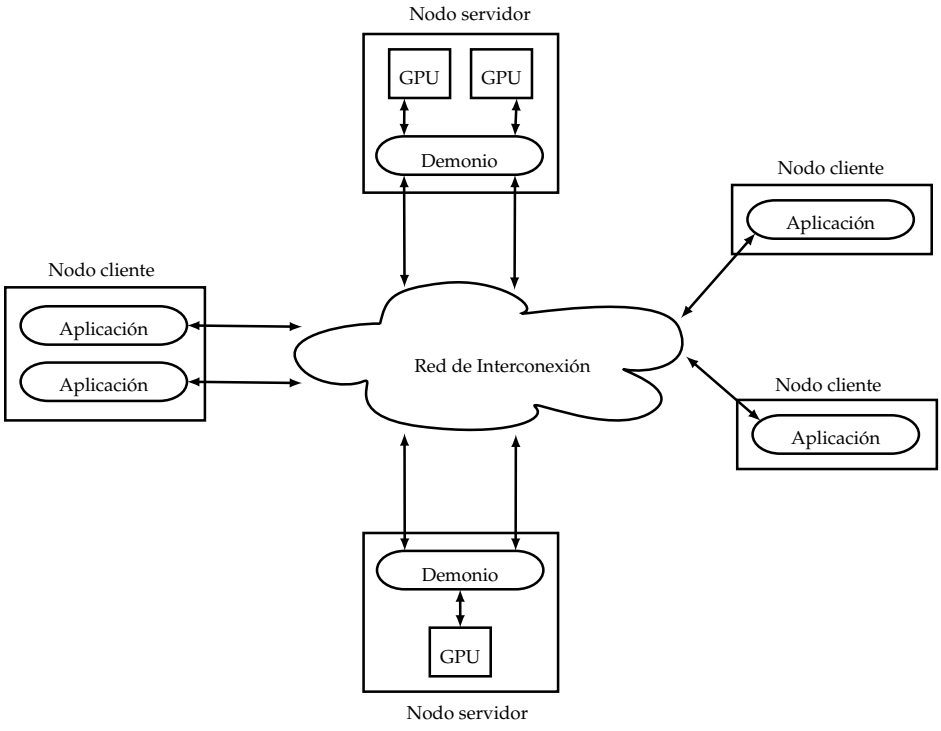


Figura 2.8: Ejemplo de la infraestructura de rCUDA sobre los nodos de un HPCC.

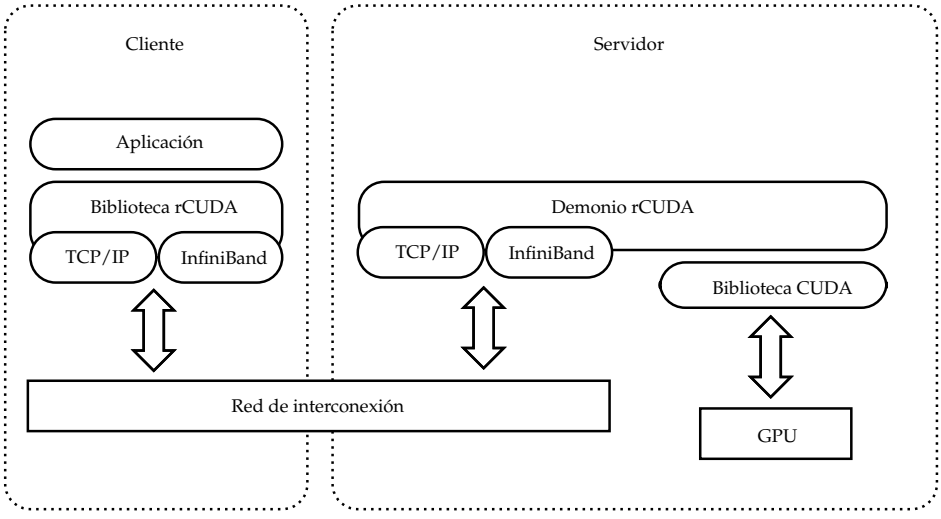


Figura 2.9: Arquitectura de rCUDA.

realizar transferencias sobre redes basadas en la familia de protocolos TCP/IP y redes IB, respectivamente [89].

Servidor

El software de servidor de rCUDA se ejecuta en los equipos que ofrecen los servicios GPGPU de una o varias GPUs. Posee las características de ser multi-instancia y *thread-safe*. Esto le confiere un alto grado de fiabilidad y le permite atender a varios clientes al mismo tiempo.

El servidor de rCUDA dispone de un demonio que permanece a la escucha de las peticiones de conexión de los clientes. Cuando se recibe una solicitud, el servidor abre una conexión con el cliente y crea un nuevo proceso que atiende la peticiones recibidas por esa conexión. De esta forma, las operaciones de CUDA solicitadas se ejecutan dentro del contexto de trabajo de ese proceso. Esto permite la utilización de la multitarea sobre la GPU, y además asegura que el fallo de un proceso no afecte al resto.

Tal y como se muestra en la Figura 2.9, al igual que ocurre con el software del cliente, el software del servidor posee los dos módulos que implementan el soporte necesario para que el servidor pueda utilizar redes TCP/IP y redes IB en las comunicaciones con el cliente.

2.6.2. Operaciones síncronas y asíncronas

Dentro de la API de CUDA se distinguen dos tipos de operaciones en función de su comportamiento: síncronas y asíncronas [16, 76, 79].

Cuando una aplicación realiza una llamada a una función síncrona de CUDA, su ejecución se bloquea hasta que la GPU completa la operación o produce un error. Esto no sucede con las operaciones asíncronas ya que el programa continúa su ejecución mientras se completa su llamada. Dentro del conjunto de servicios asíncronos de la API de CUDA se encuentran ciertas funciones de transferencia entre la memoria del equipo, o *host*, y la memoria del dispositivo, las operaciones asociadas a una hebra de CUDA (*stream*) y los lanzamientos de un núcleo de CUDA (*kernel*) [79].

Para simular este comportamiento asíncrono, el software de rCUDA utiliza hebras o hilos del SO. En concreto, como se representa en la Figura 2.10, el cliente dispone de tres hebras que le permiten llevar a cabo transferencias síncronas y asíncronas de datos [87, 91].

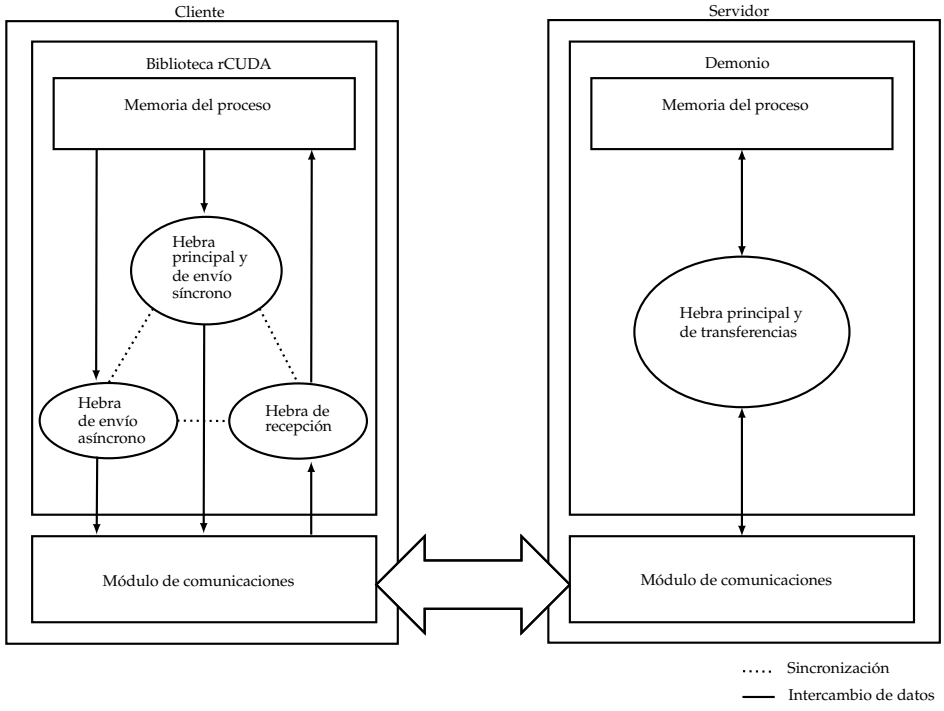


Figura 2.10: Disposición de las hebras en rCUDA.

En el caso de los envíos, cuando se realiza una llamada a una función síncrona de CUDA, la hebra principal envía la petición al servidor y a continuación, bloquea la ejecución de la aplicación, que permanece interrumpida hasta que la hebra de recepción notifica que se ha recibido el resultado. En cambio, los envíos de las operaciones asíncronas disponen de una hebra independiente que no suspende la ejecución del programa.

Con respecto a las recepciones, independientemente de si están asociadas a una operación síncrona o asíncrona, son efectuadas por una hebra dedicada que se sincroniza con la hebra de envío correspondiente.

2.6.3. Características avanzadas

Este apartado describe las técnicas que utiliza la solución rCUDA para realizar las transferencias de datos entre el cliente y el servidor de forma eficiente. Estas tecnologías son: la segmentación de transferencias (*pipeline*), la reducción del número de copias (*zero-copy*) y la eliminación de duplicados (*GPUDirect RDMA*) [19, 87, 88, 91].

Segmentación de transferencias

Como se observa en la Figura 2.11, las transferencias de datos entre el cliente y la GPU del servidor rCUDA se dividen en dos etapas. En la primera, el extremo que ejecuta el software de cliente transfiere al servidor los datos a través de la red. El servidor recibe los datos en una zona de la memoria principal y en la siguiente etapa los copia a la memoria de la GPU.

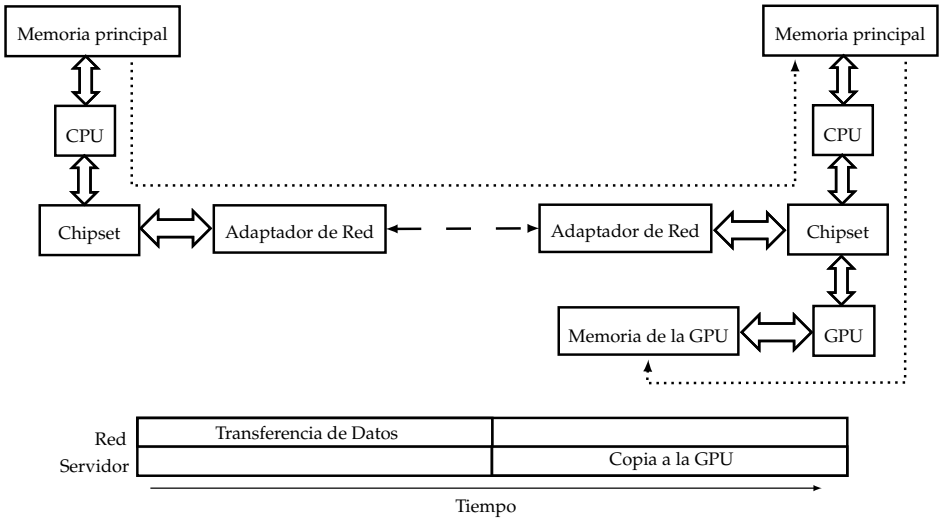


Figura 2.11: Diagrama de una transferencia de rCUDA sin segmentación.

Con el objetivo de hacer el proceso más eficiente, rCUDA utiliza la segmentación de transferencias (*pipeline*). Para ello, los datos se dividen en bloques más pequeños, y se trasmiten solapando las operaciones de transferencia con las operaciones de copia entre la memoria principal del servidor y la GPU [93]. Como el envío del siguiente bloque de datos se realiza al mismo tiempo que la copia del bloque actual a la GPU, es necesario usar varios búferes intermedios para ejecutar ambas operaciones a la vez. Concretamente se utilizan dos búferes, de forma que uno de ellos recibe el bloque de datos y el otro se emplea en la copia a la GPU. En la siguiente recepción, los búferes intercambian los papeles, de manera que el búfer que había recibido los datos los copia a la GPU [91].

En la Figura 2.12 se muestra el resultado de aplicar la segmentación a las transferencias de rCUDA. Como se observa, a pesar de añadir una

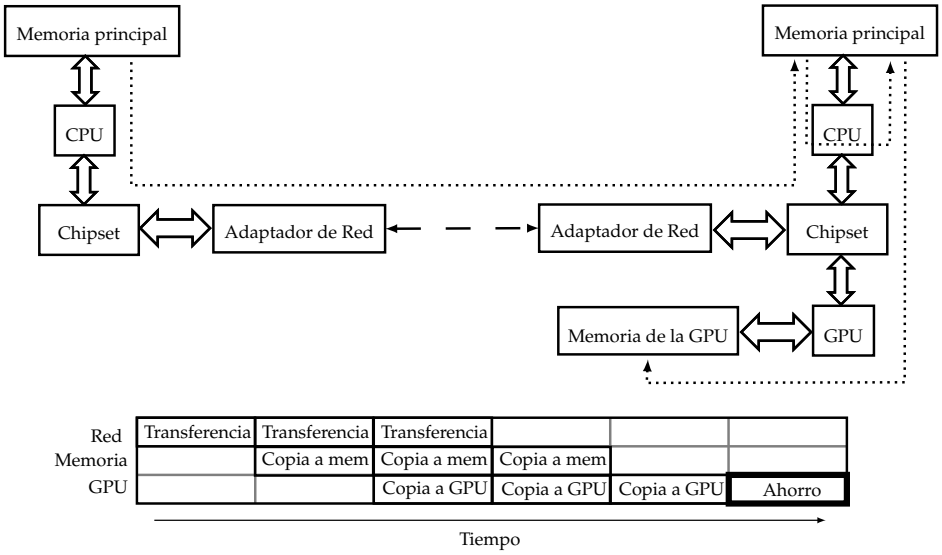


Figura 2.12: Diagrama de una transferencia de rCUDA con segmentación.

nueva etapa (la copia de los datos del búfer del controlador a uno de los búferes intermedios), con la segmentación se consigue reducir el tiempo necesario para transmitir los datos entre el cliente y el servidor.

Reducción del número de copias

Para poder aplicar esta herramienta, el SO sobre el que se ejecuta rCUDA debe soportar el bloqueo de una página de memoria. De esta forma se crea una zona de memoria no paginable que contiene el búfer donde se reciben los datos. A diferencia de lo que ocurre con la memoria paginable, al bloquear la paginación de un búfer se le indica al SO que evite mover esa porción de memoria al disco cuando libera espacio en la memoria principal (*swapping*). Esto permite que el dispositivo de red acceda directamente al búfer utilizando DMA, y que libere a la CPU de copiar los datos del búfer de recepción a los búferes intermedios [42].

En la Figura 2.13 se ilustra la reducción del número de copias en rCUDA, y cómo se elimina de este modo la etapa que añade la segmentación de transferencias (la copia de los datos desde el búfer de recepción al búfer intermedio). Esto supone un ahorro de tiempo durante las transferencias.

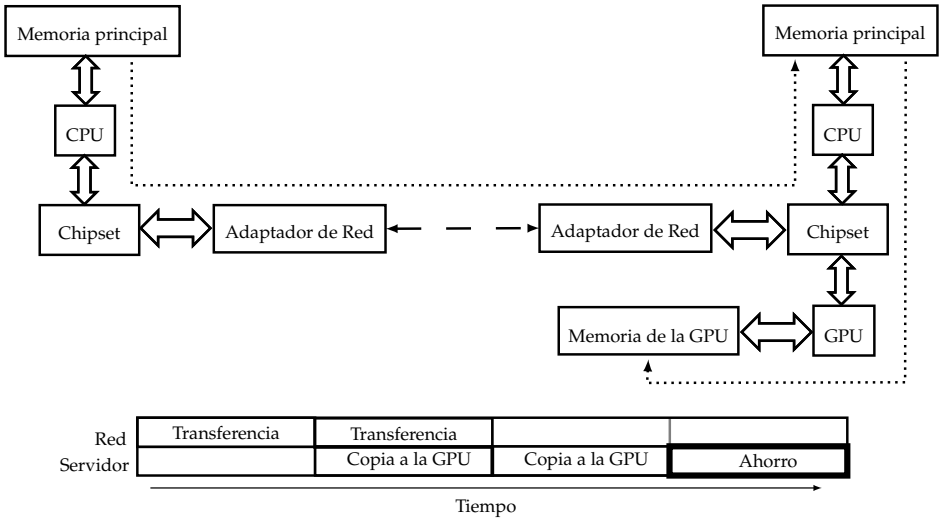


Figura 2.13: Diagrama de una transferencia de rCUDA con reducción del número de copias.

Eliminación de duplicados

Se puede reducir aún más el tiempo necesario para completar una transferencia si se suprime la copia de datos de la memoria principal del servidor a la memoria de la GPU [94]. Esto implica que la recepción de los datos se debe realizar directamente sobre la memoria de la GPU. De esta manera, como se muestra en la Figura 2.14, se ejecuta la transferencia de datos en una sola etapa. Este tipo de operaciones recibe el nombre de comunicación GPUDirect RDMA.

GPUDirect RDMA es el nombre comercial de una tecnología disponible para las GPUs del fabricante NVIDIA, que fue introducida a partir de CUDA 5.0 [16, 76, 95]. Esta técnica utiliza operaciones de DMA y requiere que el controlador del adaptador de red y el controlador de la GPU utilicen la misma zona de memoria para los datos. Esta memoria será donde el controlador del adaptador de red recibirá los datos y desde donde el controlador de la GPU los copiará a la memoria del acelerador gráfico. Para evitar que el SO traslade la memoria a disco mientras se realizan las operaciones de DMA es necesario que se marque como memoria no paginable [96].

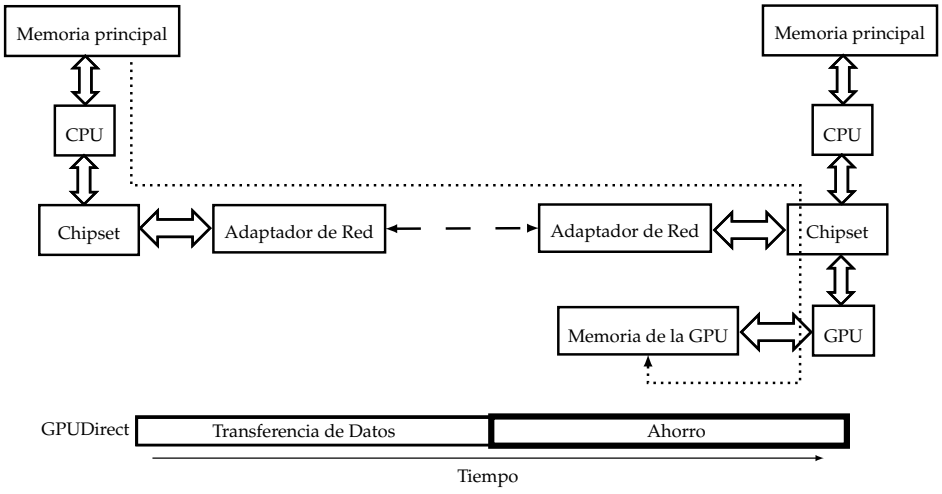


Figura 2.14: Diagrama de una transferencia de rCUDA con eliminación de duplicados.

2.7. Resumen y conclusiones

El término HPCC hace referencia a una amalgama de sistemas que van desde los grandes equipos de la lista TOP500 hasta los pequeños clústeres con unos pocos ordenadores. Sin embargo todos se caracterizan por dedicar toda su capacidad de cálculo disponible a resolver un único gran problema durante largos periodos de tiempo, como por ejemplo la predicción climática, la bioinformática, la prospección geológica o la dinámica de fluidos. Con el propósito de acortar ese tiempo, muchos HPCCs disponen de GPUs que les permiten acelerar las aplicaciones empleadas en la resolución del problema. En este capítulo se han explorado varias tecnologías orientadas a reducir el consumo energético y a maximizar el rendimiento de estos HPCCs.

En primer lugar se han presentado las dos arquitecturas de procesadores de bajo consumo que han ido ganando terreno como alternativa para mejorar la eficiencia energética de los actuales HPCCs.

En la segunda parte del capítulo se han examinado las características fundamentales de las redes de HPC. En particular, se han expuesto las ventajas que aportan las redes RDMA como sistema de interconexión de un HPCC. Frente a las redes tradicionales, RDMA define un modelo funcional

de colas y describe su especificación utilizando semánticas que reciben el nombre de "verbos". A continuación, se ha ofrecido una introducción de las APIs que permiten a las aplicaciones acceder a los servicios que proporciona RDMA. Seguidamente, se han analizado las tecnologías de red que actualmente soportan RDMA: IB, iWARP y RoCE. Estas representan el 62,4 % de los sistemas de interconexión de los HPCCs de la lista TOP500 [18].

Para finalizar, se ha realizado un estudio comparativo de las tecnologías de virtualización de GPUs orientadas a GPGPU, donde destaca la solución ofrecida por rCUDA. Posteriormente se ha explorado el funcionamiento del modelo cliente-servidor propuesto por el software de rCUDA, así como los mecanismos utilizados en las transferencias sobre las tecnologías de interconexión IB y redes TCP/IP.

CAPÍTULO 3

Módulo de comunicaciones de rCUDA para tecnologías RDMA

La estructura de rCUDA incorpora dos módulos de comunicaciones que implementan las funcionalidades y operaciones necesarias para la interconexión de los extremos y la transmisión de datos. Hasta la fecha, sólo se dispone de los módulos de comunicaciones para redes basadas en TCP/IP y para redes IB. Por tanto, se debe utilizar el módulo TCP para intercambiar datos sobre infraestructuras de red RDMA que no sean del tipo IB. En estos casos, el rendimiento de las comunicaciones se encuentra penalizado por la sobrecarga inherente a TCP/IP y el desaprovechamiento de las capacidades que ofrecen las tecnologías RDMA. Esto, por ejemplo, supone un gran inconveniente cuando se ejecuta rCUDA en los HPCCs que emplean infraestructuras de red 10GbE con soporte para RDMA.

Para superar este contratiempo se propone la ampliación del software de rCUDA con el sistema de comunicaciones desarrollado en el presente trabajo, que mejora el rendimiento para las tecnologías RDMA. De esta forma, se dota a rCUDA del potencial para establecer configuraciones donde las aplicaciones científicas paralelas acceden a GPUs remotas a través de las infraestructuras de red más habituales de la lista TOP500 [18].

En este capítulo se exponen las tres alternativas confeccionadas en la presente investigación con el objetivo de permitir a rCUDA alcanzar rendimientos sobre redes RDMA mayores a los que obtiene con el módulo TCP. En la Sección 3.2 se presenta y evalúa la primera de las implementaciones, que aporta una aproximación inicial, por lo que carece de algunas funcionalidades. En las Secciones 3.4 y 3.5, se ofrece una descripción detallada de los mecanismos de conexión y transmisión de los dos módulos que forman la solución completa. Posteriormente, en la Sección 3.6, se profundiza en los ajustes y los parámetros avanzados de estos módulos. En la Sección 3.7 se realiza un análisis comparativo del rendimiento de todos los módulos de rCUDA. Por último, las conclusiones del capítulo se exponen en la Sección 3.8.

3.1. Arquitectura

Como se indicó en el Apartado 2.6.1, el software de rCUDA se divide en dos componentes: cliente y servidor. Cada uno de los componentes ofrece unas operaciones de conexión y comunicación diferenciadas donde, por ejemplo, un servidor rCUDA permanece a la espera de recibir las peticiones de un cliente. Esto requiere que el módulo de comunicaciones disponga de artefactos específicos para cada uno de los componentes.

Además, como se explicó en el Apartado 2.6.1, dentro del servidor se distinguen dos tipos de procesos participantes. El primero de ellos se encarga de recibir la petición de conexión de un cliente rCUDA, inicializa la comunicación y cede el control al otro participante. Este segundo proceso es responsable de ejecutar todas las operaciones CUDA y devolver los resultados al cliente.

Después de lo expuesto, para dar respuesta a estas funcionalidades, los módulos de comunicación desarrollados en el presente trabajo disponen de las siguientes entidades:

`Solicitante`. El cliente rCUDA utiliza esta entidad para comunicarse con un servidor rCUDA y realizar peticiones. En concreto, el cliente crea una entidad `Solicitante` por cada GPU remota a la que accede.

`Oyente`. Está disponible en el servidor rCUDA y se genera cuando éste se pone en funcionamiento. Sólo existe una instancia de la entidad `Oyente` por servidor. Esta instancia espera las peticiones de conexión de los clientes. Al recibir una solicitud, inicializa una entidad `Agente` y le transfiere el control de la conexión. El `Oyente` nunca envía ni recibe datos del cliente.

`Agente`. Cuando un servidor rCUDA recibe una petición de conexión, se crea un `Agente` y se le asigna la tarea de recibir las peticiones del extremo remoto. De esta forma el servidor dispone de tantas entidades `Agente` como conexiones abiertas con clientes existen. El `Agente` es responsable de reservar la memoria necesaria para recibir las peticiones, crear las estructuras de datos y de control, y llevar a cabo todas las transferencias de datos con un `Solicitante` del cliente. Finalmente, cuando el `Solicitante` cierra la conexión, su `Agente` asociado libera los recursos reservados y finaliza.

3.2. Aproximación inicial al módulo de comunicaciones

En esta sección se discute el diseño e implementación de la primera aproximación del módulo de comunicación de rCUDA sobre redes RDMA, llamado FAR (*First Approach over RDMA*). El objetivo que se persigue es demostrar que es posible tener un módulo para rCUDA capaz de mejorar el rendimiento que se obtiene con el módulo TCP sobre todas las tecnologías RDMA existentes sin llegar a ser un desarrollo completo.

3.2.1. Mecanismo de transmisión

El módulo FAR implementa un mecanismo de transmisión entre los extremos que se caracteriza por utilizar operaciones de semántica de canal, establecer un doble canal de comunicación, registrar todos los búferes

usados en las transferencias y agrupar los envíos en ráfagas. En los siguientes puntos de este apartado se describe cada uno de estos conceptos detalladamente.

Semántica de comunicación

Como se discutió en el Apartado 2.6.1, la arquitectura de rCUDA se compone de un servidor que espera peticiones; y unos clientes, que después de realizar una solicitud, aguardan a que el servidor les devuelva los resultados. Por eso, resulta muy sencillo utilizar operaciones de semántica de canal de la API para la implementación del módulo FAR, ya que estas operaciones requieren la participación de ambos extremos. De esta manera el emisor envía un mensaje mientras que el receptor espera la llegada de los datos.

Doble canal de comunicación

Generalmente el cliente rCUDA envía una petición al servidor con la información de la función que se desea ejecutar. Además, en los casos que corresponda, realiza otra transferencia con los datos de entrada de la función.

En el otro lado, el servidor recibe la información de la función y comprueba su identificador para determinar si es necesario publicar una solicitud de recepción adicional para recibir los datos de entrada de la función. En estos casos, puede ocurrir que el cliente realice el envío de los datos antes de que se publique la recepción, lo que produce un estado de error en el par de colas o QPs (*Queue Pair*s) del mecanismo de comunicación de RDMA.

Como consecuencia, se produce una pérdida rendimiento motivada por la necesidad de restablecer el estado del QPs [43] y la retransmisión de la última petición del cliente para retomar la ejecución de la aplicación.

Este problema también puede presentarse en el extremo del cliente, ya que después de recibir el resultado de una petición, debe decidir si es necesario publicar una nueva solicitud de recepción para los datos de salida de la función.

Una solución habitual es mantener siempre publicadas solicitudes de recepción en el extremo en el que se esperan los datos [41, 44, 46, 47].

Sin embargo, esta técnica no es válida para rCUDA, ya a que a priori no es posible saber cuál es el tamaño de los datos que se transferirán. Esto depende de cada función y, en algunos casos, de cada solicitud. Puede ocurrir que una función no necesite datos adicionales, o que incluso requiera muchos menos que la cantidad de recepciones publicadas. Entonces se presenta otro grave problema, ya que una solicitud de trabajo o WR (*Work Request*) publicada no puede ser retirada.

Para hacer frente a esta situación, y obtener un buen rendimiento, el módulo FAR crea dos canales de comunicación, donde cada uno de ellos usa un par de colas o QP, y una cola de finalización o CQ (*Completion Queue*) independientes. En cada canal, los datos y la información fluyen en una dirección mientras que en sentido opuesto se envían los mensajes de control de flujo. El receptor emplea estos mensajes con la finalidad de notificar al emisor que ha publicado las solicitudes de recepción necesarias para acoger los datos de las transferencias. Como resultado, la recepción de uno de estos mensajes puede considerarse la señal de partida para que el emisor comience el proceso de publicación de las solicitudes de envío.

En la Figura 3.1 se muestra el proceso que ejecuta el módulo FAR para establecer el doble canal entre los extremos. En concreto, los pasos que se realizan son los siguientes:

1. En primer lugar, el servidor crea la entidad `Oyente`, cuya tarea es permanecer a la escucha de peticiones de conexión de los clientes.

2. En el otro extremo, la entidad `Solicitante` del cliente publica una solicitud de recepción para almacenar el número de puerto del segundo canal, que será proporcionado por el servidor. Posteriormente, como puede observarse en el paso 2 del diagrama de la Figura 3.1, el `Solicitante` realiza una petición de conexión al servidor. El cliente permanece a la espera hasta que se abre el canal de comunicación con el servidor y se recibe el número de puerto donde debe realizar la petición de conexión del segundo canal.

3. Cuando se recibe la solicitud de conexión del `Solicitante`, el `Oyente` crea una entidad `Agente` y le traspasa a éste la solicitud. Como se muestra en el paso 3 del diagrama de la Figura 3.1, la entidad

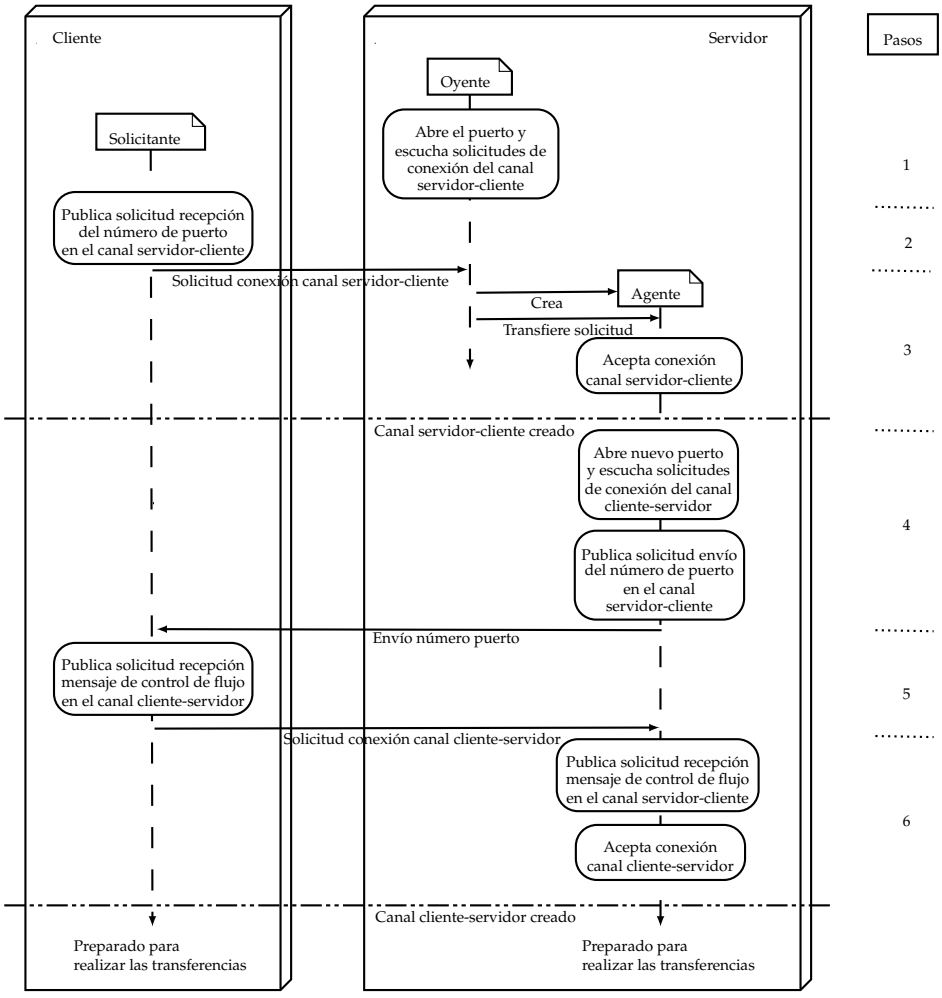


Figura 3.1: Diagrama del proceso de conexión del módulo FAR.

`Agente` lleva a cabo las siguientes acciones para atender la solicitud recibida del `Solicitante`:

a) Obtiene el identificador de conexión.

b) Crea las estructuras de datos necesarias para la conexión entre el servidor y el cliente.

c) Acepta la conexión y abre el canal servidor-cliente.

d) Reserva un nuevo puerto para recibir la petición de conexión del segundo canal de comunicación y permanece a la escucha.

e) Envía la dirección del nuevo puerto al `Solicitante` a través del canal servidor-cliente.

4. Después, el cliente publica la solicitud de recepción de los mensajes de control de flujo en el canal que transportará los datos desde el cliente al servidor, denominado canal cliente-servidor. A continuación, realiza una nueva petición de conexión al servidor dirigida al segundo puerto y espera hasta que se abre el segundo canal de comunicación.

5. Cuando la entidad `Agente` recibe la segunda solicitud de conexión del `Solicitante`, ejecuta las tareas correspondientes al paso 6 del diagrama de la Figura 3.1:

   a) Obtiene el identificador de la nueva conexión.

   b) Genera las estructuras de datos necesarias para la conexión entre el cliente y el servidor.

   c) Publica la solicitud de recepción de los mensajes de control de flujo en el canal servidor-cliente.

   d) Acepta la conexión y abre el canal de comunicación encargado de transportar los datos desde el cliente hasta el servidor (canal cliente-servidor).

6. A partir de este momento, se pueden ejecutar las transferencias de datos entre los sistemas cliente y servidor.

Registro de los búferes de las capas superiores

En el Apartado 2.3.1 se indicó que el primer paso para poder realizar una transferencia sobre una tecnología RDMA consiste en registrar los búferes que se utilizarán en el envío y en la recepción de los datos. El módulo FAR ofrece a las capas superiores del software de rCUDA la posibilidad de registrar sus propios búferes internos. Esto permite que las operaciones de transferencia se realicen directamente sobre esas zonas de memoria sin necesidad de utilizar búferes intermedios. Con esto se pretende evitar la pérdida de rendimiento que aparece al realizar la copia de datos entre los búferes del módulo de comunicación y de las capas superiores. Además,

si el búfer supera cierto tamaño, su registro se divide en porciones que pueden ser manipuladas de forma más eficiente por el *pipeline* de rCUDA, como se explicó en el Apartado 2.6.3.

Agrupación de las transferencias

Como se describió en el Apartado 3.2.1, para poder sincronizar las comunicaciones entre el cliente y el servidor se utilizan mensajes de control de flujo sobre un doble canal de comunicación. Pero si se aplica este mecanismo para cada transferencia, es decir, si se envía un mensaje de control de flujo por cada envío de datos, se reduce el ancho de banda efectivo. Por ejemplo, para 10 paquetes de datos son necesarios 10 mensajes de control, lo que supone que la mitad de las operaciones de envío corresponden a mensajes de control.

Para minimizar el impacto de los mensajes de control de flujo en el rendimiento, el módulo FAR agrupa varios envíos en una ráfaga. Cada ráfaga está formada por un número de transferencias cuyo límite es la cantidad máxima de WRs que cada tecnología RDMA permite encolar o publicar en los pares de colas.

Como se observa en la Figura 3.2, la transmisión de una ráfaga se inicia cuando el emisor de los datos recibe un mensaje de control de flujo. Al finalizar, ambos extremos comprueban que todas las transferencias que agrupa la ráfaga se han completado correctamente. Después, el remitente queda a la espera de recibir un nuevo mensaje de control de flujo para procesar la siguiente ráfaga. Mientras, el destinatario publica las solicitudes de recepción. Cuando termina, envía un mensaje de control de flujo al remitente para que comience la nueva ráfaga. Así pues, el protocolo queda de la siguiente manera:

1. Antes de iniciar el intercambio de datos, el emisor publica una solicitud de recepción de un mensaje de control de flujo, que se añade a las ya existentes. De esta forma se asegura la existencia de solicitudes de recepción para los futuros mensajes de control de flujo. Después, el emisor espera hasta que una solicitud de recepción de un mensaje de control de flujo se complete (paso 1 del diagrama de la Figura 3.2).

2. En el otro extremo, el receptor publica las solicitudes de recepción

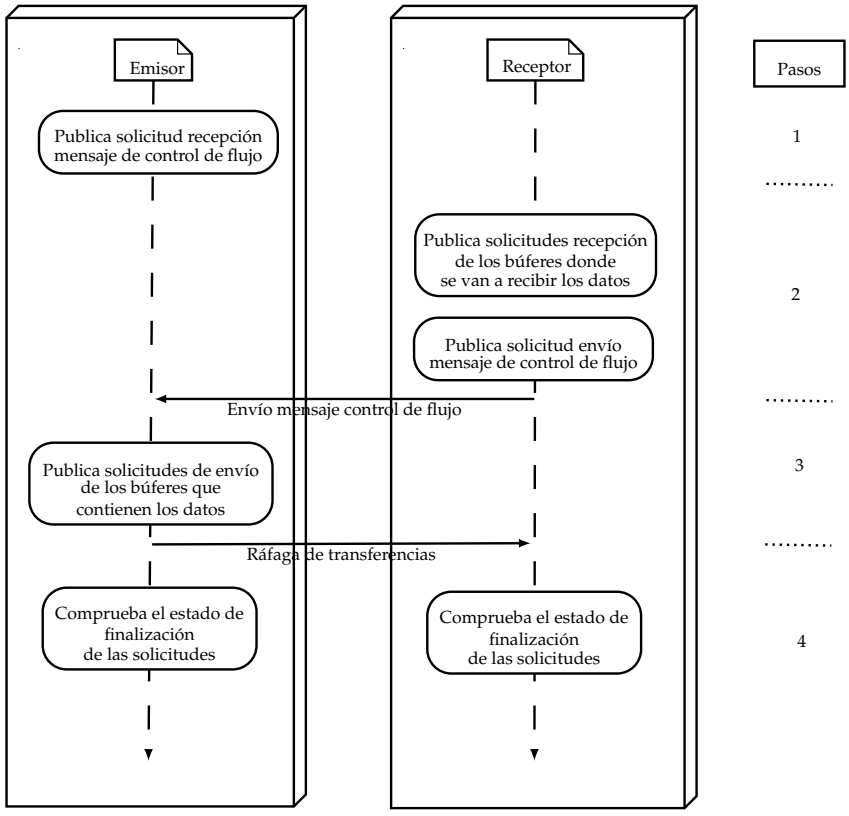


Figura 3.2: Diagrama del proceso de transmisión del módulo FAR.

necesarias para almacenar los datos que se van a transferir. Posteriormente, tal y como se muestra en el paso 2 del diagrama de la Figura 3.2, envía un mensaje de control de flujo para indicar al emisor que las solicitudes de recepción de los datos han sido publicadas.

3. Al recibir este mensaje, el emisor comienza la publicación de las solicitudes de envío de la ráfaga de transferencias.

4. Cuando las solicitudes de envío y recepción se completan, lo que se corresponde con el paso 4 del diagrama de la Figura 3.2, el emisor y el receptor comprueban el estado de finalización de cada una de ellas.

5. Por último, en el caso de que existan datos pendientes de transmitir, se repite el proceso desde el principio.

3.2.2. Desarrollo inicial

Se debe entender que el módulo FAR ha sido desarrollado con la finalidad de ser el punto de partida para las futuras soluciones productivas que efectúan las comunicaciones entre los clientes y los servidores de rCUDA sobre todas las tecnologías RDMA. Por esta razón no implementa la funcionalidad completa de un módulo de comunicaciones. En su lugar, únicamente dispone de la operativa básica para que rCUDA proporcione los servicios de virtualización que requiere la aplicación encargada de evaluar el potencial de la aproximación. En otras palabras, aquellos servicios que permiten explorar la viabilidad de los mecanismos utilizados y obtener los resultados necesarios para realizar una comparativa con los módulos de comunicación de rCUDA existentes.

En concreto, el módulo FAR presenta las siguientes carencias:

- El desarrollo no ha sido preparado para atender peticiones simultáneas de varios clientes. Es decir, un servidor sólo puede mantener la comunicación con un cliente. Por tanto, para abrir una nueva conexión se debe cerrar la existente.

- La gestión de *streams* de CUDA no se ha implementado.

- Las operaciones asíncronas CUDA no están soportadas. De manera que cada operación ejecutada presenta un comportamiento síncrono, independientemente de sus especificaciones.

- Por último, se limita la funcionalidad de reiniciar el dispositivo CUDA (operación `cudaDeviceReset`), ya que esto requiere una gestión avanzada de las comunicaciones que queda fuera del alcance de las funciones básicas que debe ofrecer la aproximación inicial. Además, no es una funcionalidad requerida por la aplicación empleada en la evaluación de la aproximación.

3.2.3. Evaluación de la aproximación

En este apartado se analiza el rendimiento de rCUDA al utilizar el módulo FAR para efectuar las comunicaciones entre los clientes y los servidores a través de las tecnologías RDMA presentadas en la Sección 2.4.

Para realizar la evaluación experimental se ha empleado la siguiente infraestructura:

SVR. Es un multiprocesador de memoria compartida que dispone de un procesador Intel Xeon E5-2630L a 2,00 GHz y 8 GB de RAM.

El ordenador está conectado a una red RDMA a través de una tarjeta que se ubica en un puerto de expansión PCIe 3.0 x8.

En otro puerto PCIe 3.0 x16 se encuentra ensamblada una GPU NVIDIA GeForce GTX 480 "Fermi" con 448 núcleos.

CLI. Esta máquina está equipada con un procesador Intel Core 2 Quad modelo X3430 (2,40 GHz), con 8 GB de RAM y un dispositivo de red RDMA conectado a través de un puerto PCIe 2.0 x8.

El SO utilizado en ambas plataformas es Linux CentOS 6.6 con kernel 2.6.32-50. Además, ambos equipos incorporan el compilador GNU gcc/g++ 4.4.7, la API de CUDA 6.0 y la API de OFED 3.12-1-rc4.

Para que las máquinas `CLI` y `SVR` accedan a las infraestructuras de red RDMA, se instala, según convenga, uno de los siguientes adaptadores en el puerto PCIe que se ha indicado anteriormente:

IB. La conexión a la red IB se realiza a través de un dispositivo Mellanox MCX353A-QCBT para PCIe 3.0 x8 que implementa el estándar QDR (*Quad Data Rate*).

iWARP. El acceso a la red iWARP se efectúa mediante una tarjeta NetEffect NE020 que soporta PCIe 1.1 x8.

RoCE. Para la interconexión RoCE se dispone de un adaptador Mellanox MCX312A-XCBT con capacidad PCIe 3.0 x8.

Con el propósito de evaluar el rendimiento de los módulos de comunicaciones de rCUDA sobre estas infraestructuras de red RDMA, se ha empleado la herramienta `bandwidthTest`, que se encuentra disponible en el kit de desarrollo de CUDA 6.0. [81]. Esta aplicación mide el ancho de banda de las transferencias entre un *host* y un dispositivo CUDA cuando se utiliza memoria paginable o memoria no paginable.

Los experimentos han consistido en la ejecución de `bandwidthTest` de modo que efectúa transferencias a una GPU virtualizada a través de las

infraestructuras de red RDMA. Para ello se ha instalado el servidor rCUDA en la máquina `SVR`, mientras que el cliente rCUDA y la aplicación se han ubicado en `CLI`.

En las Figuras 3.3, 3.4 y 3.5 se representan los resultados obtenidos por `bandwidthTest` cuando se emplean los módulos TCP, IB y FAR para llevar a cabo las comunicaciones entre el cliente y el servidor rCUDA sobre redes RDMA.

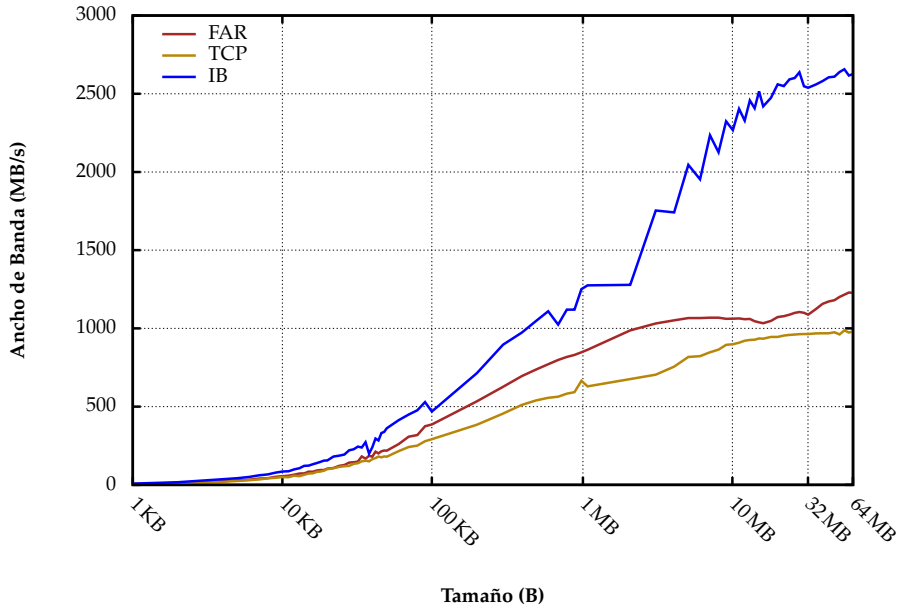


Figura 3.3: Evaluación del rendimiento del módulo FAR sobre IB.

Para la infraestructura IB (Figura 3.3), se observa que el ancho de banda obtenido por el módulo FAR supera al proporcionado por el módulo TCP. Sin embargo, sigue muy alejado de los valores alcanzados con el módulo IB. Por ejemplo, para tamaños de 100 KB el módulo IB ofrece un 20 % más de ancho de banda y para 64 MB la diferencia se amplia hasta el 114 %.

Por otro lado, cuando se examinan las Figuras 3.4 y 3.5, puede verse como el rendimiento de la aproximación inicial está cercano al rendimiento del módulo TCP. En concreto, la mejora del rendimiento del módulo TCP frente a FAR no es mayor al 19,7 % en RoCE. Mientras que en el caso de iWARP, el módulo FAR consigue sobrepasar hasta en un 7 % el ancho de banda del módulo TCP para tamaños entre 300 KB y 16 MB.

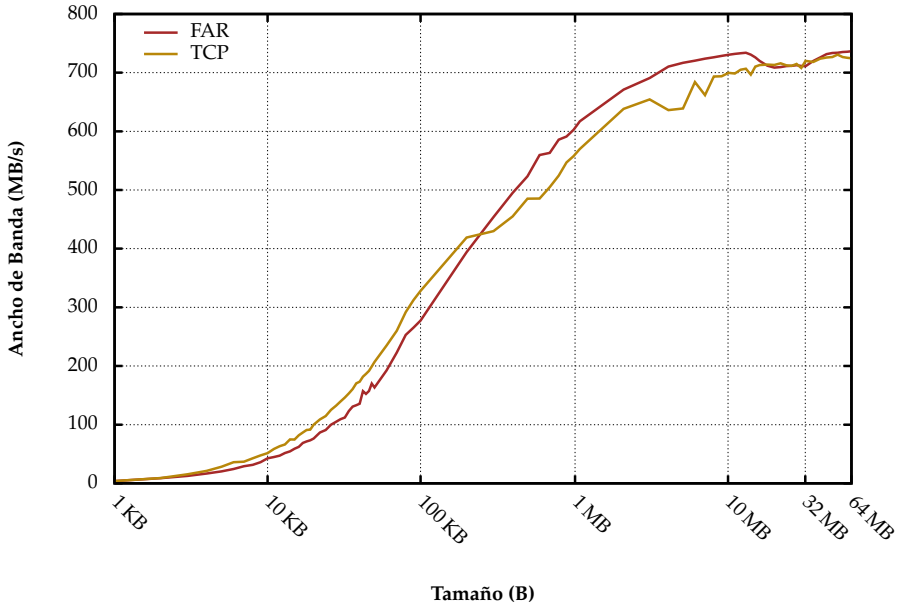


Figura 3.4: Evaluación del rendimiento del módulo FAR sobre iWARP.

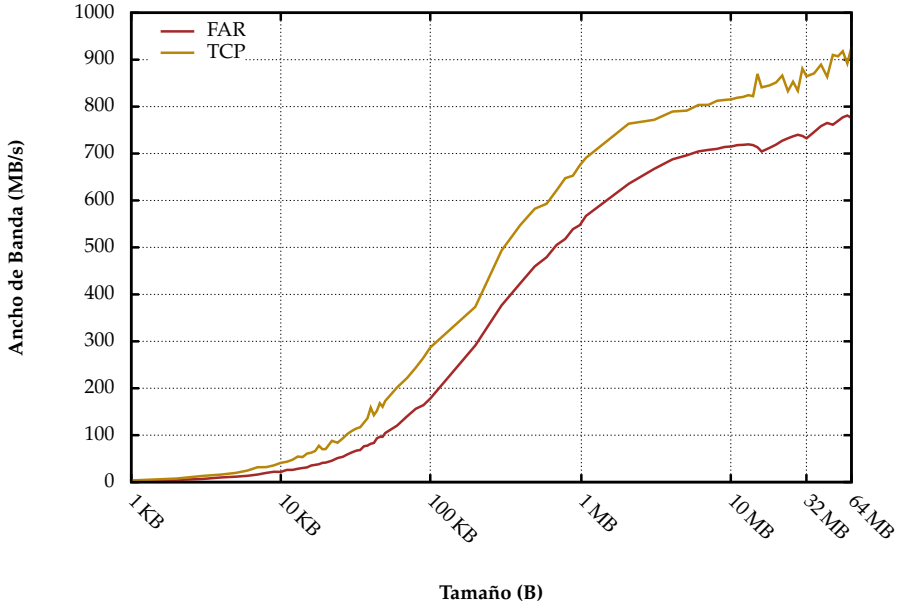


Figura 3.5: Evaluación del rendimiento del módulo FAR sobre RoCE.

El hecho de que el rendimiento del módulo TCP se sitúe por encima del obtenido por la aproximación inicial es consecuencia principalmente de dos factores:

1. Los dispositivos de conexión RoCE y iWARP disponen de mecanismos que minimizan la sobrecarga que introduce el protocolo TCP/IP.

2. El protocolo de comunicación implementado en la aproximación no aprovecha todo el potencial que ofrecen estas tecnologías, como por ejemplo las operaciones de semántica de memoria, el mecanismo GPUDirect RDMA o el envío de mensajes sin datos.

En definitiva, los resultados expuestos en las Figuras 3.3, 3.4 y 3.5 indican que es posible desarrollar un módulo de comunicaciones para rCUDA capaz de operar sobre las tecnologías RDMA y explotar sus ventajas para superar el rendimiento obtenido con el módulo TCP. En las siguientes secciones se describen las soluciones diseñadas para alcanzar esta meta.

3.3. Aspectos generales de las soluciones completas

En la sección anterior se presentó y evaluó un prototipo del módulo de comunicaciones de rCUDA para las redes RDMA. Esta aproximación inicial es capaz de funcionar sobre cualquier tecnología RDMA, y como se ha demostrado, puede superar el rendimiento del módulo TCP. Sin embargo, es necesario perfeccionar los mecanismos de transmisión utilizados con el fin de alcanzar el máximo rendimiento posible en las tecnologías RDMA.

La API `rdmacm` ofrece dos conjuntos de operaciones que pueden ser utilizadas para implementar mejoras en las transmisiones: las operaciones de semántica de canal y las operaciones de semántica de memoria.

En el Apartado 3.2.1 se explicó cómo las operaciones de semántica de canal se ajustan al funcionamiento del modelo cliente-servidor de rCUDA, donde los extremos utilizan operaciones de envío y recepción.

No obstante, las operaciones de semántica de memoria permiten reducir el trabajo que ejecuta el extremo pasivo de la comunicación. Así, una transferencia puede ser ejecutada sin la necesidad de la intervención del receptor. Es decir, no se requiere ninguna acción en el extremo receptor.

Como se ha descrito, ambas semánticas ofrecen beneficios. Sin embargo es difícil afirmar cuál de las dos puede hacer que rCUDA obtenga mayor rendimiento sobre redes RDMA. Así pues, se ha decidido desarrollar dos soluciones productivas, donde cada una de ellas utiliza únicamente las operaciones de una de las dos semánticas de comunicación para las transferencias de datos.

En las siguientes secciones de este capítulo se describen de forma detallada los dos desarrollos completos del módulo de comunicaciones de rCUDA para tecnologías de red con soporte para RDMA. Como se verá, ambos toman como punto de partida el trabajo realizado en la aproximación inicial.

3.4. Módulo de comunicaciones basado en semántica de canal

En esta sección se presenta la primera de las dos soluciones completas generadas a partir del módulo FAR, cuyo nombre es CSR (*Channel Semantics over RDMA*). En particular, este módulo emplea únicamente operaciones de semántica de canal para realizar las transferencias de datos. A continuación se exponen los elementos clave de su diseño e implementación.

3.4.1. Creación de dos canales de comunicación

Del mismo modo que ocurre con el prototipo que se presentó en el Apartado 3.2.1, el módulo CSR recurre a un doble canal de transmisión. Sin embargo, en este caso se establecen los dos siguientes canales dedicados:

Canal de datos. Los extremos de la comunicación emplean este canal para el intercambio de la información y los datos de las peticiones que realiza la aplicación a la GPU remota. Por ello es conveniente que disponga de un elevado ancho de banda.

Canal de control. Por este canal se emiten los mensajes de control de flujo de los envíos y recepciones del canal de datos. Para que las transferencias por el canal de control consuman el menor ancho de banda posible, se utilizan mensajes cuyo campo de datos está vacío.

3.4.2. Mecanismos utilizados en la transferencia de datos

En este apartado se analizan los mecanismos implementados en el módulo CSR con el propósito de incrementar el rendimiento de las transferencias.

Empleo de búferes intermedios para las comunicaciones

Como ya se describió en el Apartado 2.3.1, antes de ejecutar una operación RDMA es necesario registrar la región de memoria donde se almacenan los búferes usados en los envíos y recepciones. Al inicio del proceso de registro, el controlador del adaptador de canal empareja la dirección de memoria virtual con la dirección de memoria física. A continuación, el controlador bloquea la región de memoria para asegurarse de que el sistema operativo no enviará esa memoria al espacio de intercambio mientras se realizan las operaciones RDMA. Sin embargo, una región de memoria no puede ser inmovilizada para siempre, ya que el tamaño efectivo de la memoria física utilizada para otros fines se reduce. Por otro lado, el número de entradas en la tabla que mantiene la información de todas las regiones de memoria registradas es limitado. Cuando el número de búferes registrados supera este límite, o se requiere la memoria que está bloqueada, la aplicación debe deshacer el registro y liberar recursos en el adaptador de canal. Como se documenta en [97, 98], el registro de memoria y su destrucción son operaciones costosas. En particular, en los estudios [97, 99] se demuestra que el exceso de registros de memoria y sus anulaciones penalizan el rendimiento de las aplicaciones RDMA. En consecuencia, bajo ciertas condiciones, la copia tradicional de los datos a un búfer intermedio y su envío mediante una operación RDMA puede proporcionar un mayor rendimiento.

Debido a esto, y a diferencia del desarrollo FAR, el módulo CSR no permite que las capas superiores del software de rCUDA registren sus búferes para posteriormente utilizarlos en las comunicaciones RDMA. En su lugar, el módulo CSR utiliza dos búferes intermedios de idéntico tamaño, el búfer de envío y el búfer de recepción, que se dividen en un conjunto de porciones de dimensión fija, donde cada una de estas divisiones es una zona de memoria registrada independiente. Al iniciarse el software de virtualización de rCUDA, ambos extremos de la comunicación publican

una solicitud de recepción asociada a cada porción del búfer de recepción en el canal de datos. Cuando se notifica la llegada de un mensaje, se comprueba que no existen errores, se extraen los datos recibidos, y se publica una nueva solicitud de recepción para esa porción del búfer. De esta forma siempre existen solicitudes de recepción esperando los envíos del extremo opuesto.

Agrupación de transferencias en ráfagas

La división de los búferes intermedios en porciones facilita la utilización de la técnica de agrupación de transferencias en ráfagas que ha sido presentada en el desarrollo de la aproximación inicial. El proceso sería el siguiente:

1. El emisor emite una solicitud de recepción en el canal de control de flujo.

2. Posteriormente, el emisor publica en el canal de datos las solicitudes de envío de las porciones del búfer intermedio de envío hasta completar una ráfaga.

3. A continuación, el emisor espera a que el receptor procese los envíos de la ráfaga y publique nuevas solicitudes de recepción.

4. Cuando el receptor finaliza, envía al emisor un mensaje de control de flujo a modo de notificación.

5. En el instante en que el emisor recibe el mensaje, continúa la ejecución y repite el proceso hasta que se han enviado todos los datos restantes.

Notificación de recepciones con antelación

Con la intención de solapar las etapas de envío de mensajes de control de flujo con la recepción de datos, se implementa un mecanismo para la notificación de las recepciones de datos con antelación. De esta forma el receptor envía el mensaje de control cuando, después de haber procesado parte de las solicitudes de recepción de una ráfaga, el número de recepciones publicadas puede atender todos los envíos de una nueva ráfaga. Para ello es necesario que el número de transferencias que contiene una ráfaga

sea menor que el número de porciones de los búferes de envío y recepción. Por otro lado, para poder utilizar la implementación del *pipeline* de rCUDA con la notificación con antelación proporcionada por el módulo CSR, es necesario que el número de transferencias de la ráfaga sea múltiplo de 3 y mayor o igual a 6. Como resultado, las ráfagas usadas en el módulo CSR están formadas por $n - 3$ transferencias con $n \geq 9$ y múltiplo de 3, donde n es el número de porciones de los búferes.

Comprobación tardía de errores en la comunicación

Otra herramienta que se aplica con el objetivo de aumentar la velocidad de las transferencias del módulo CSR es la comprobación tardía de errores. La API `rdmacm` dispone de un mecanismo de acarreo de errores que almacena la información del error hasta el momento en que es consultada. Por tanto, el emisor solamente debe comprobar si han ocurrido errores en la comunicación al finalizar el último envío de una ráfaga, ya que, si se ha producido un error en cualquier envío anterior, este todavía estará registrado.

Esto reduce el tiempo que se dedica a la verificación del transporte de datos, que puede ser reasignado a realizar trabajo efectivo para la aplicación.

Utilización de GPUDirect RDMA en las transferencias

Finalmente, se incorpora el mecanismo de GPUDirect RDMA descrito en el Apartado 2.6.3 al desarrollo. Con ello se minimiza el tiempo necesario para trasladar los datos de los búferes intermedios del módulo de comunicación a la zona definitiva de la memoria de la GPU.

3.4.3. Proceso de establecimiento de conexión

Una de las limitaciones del módulo FAR desarrollado en el presente trabajo es que no puede atender peticiones simultáneas de distintos clientes. Esto se debe a que algunas de las funciones de la API `rdmacm` utilizadas no son *thread-safe*. En consecuencia, se puede producir un error cuando las entidades `Agente` de los procesos creados por el demonio del servidor rCUDA (véase el Apartado 2.6.1) comparten las estructuras de datos generadas por las funciones de la API.

Para superar este obstáculo, el protocolo de establecimiento de comunicación utiliza los sockets POSIX para recibir las peticiones de conexión. De esta forma el `Oyente` no necesita realizar llamadas a la API `rdmacm`, sino que todas las llamadas son realizadas por la entidad `Agente` y son ejecutadas dentro del contexto de cada subproceso. Así se evita que los subprocesos compartan estructuras de datos generadas por la API.

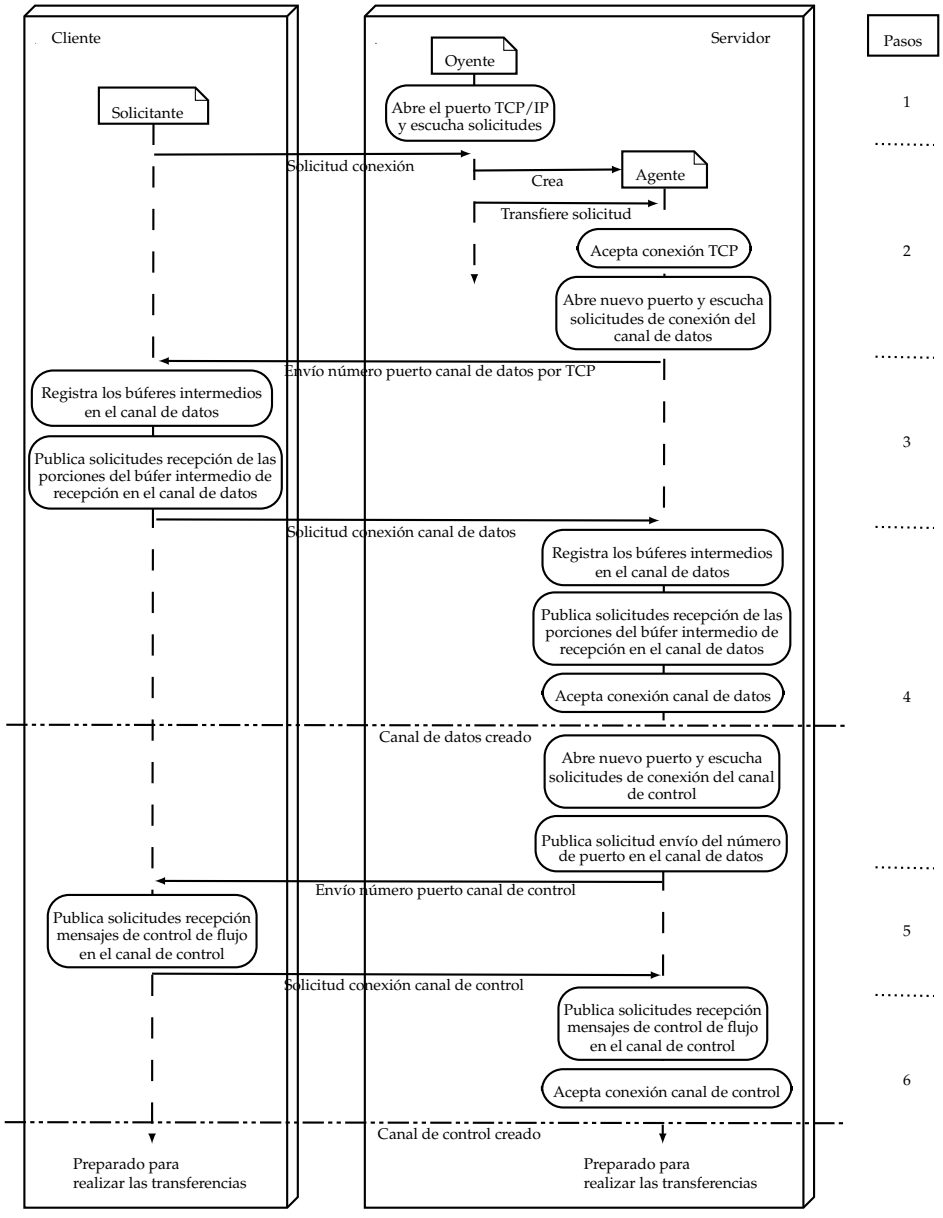


Figura 3.6: Diagrama del establecimiento de conexión del módulo CSR.

En la Figura 3.6 se detalla el procedimiento que realiza el módulo CSR para establecer la comunicación entre el cliente y el servidor de rCUDA. A continuación se describe la secuencia de pasos que ejecuta cada extremo.

Secuencia de pasos del cliente

1. Como se muestra en la Figura 3.6, la entidad `Solicitante` inicia el proceso de establecimiento de la comunicación con el envío de una petición de conexión TCP/IP al servidor.

2. El cliente se detiene hasta que se abre la conexión TCP/IP y recibe el número de puerto donde debe enviar las peticiones de conexión para el canal de datos.

3. Una vez se recibe ese dato, como puede observarse en el paso 3 del diagrama de la Figura 3.6, el `Solicitante` lleva a cabo las siguientes acciones:

   a) Crea las estructuras de datos necesarias para la comunicación.

   b) Registra los búferes intermedios utilizados para el envío y la recepción del canal de datos.

   c) Publica las solicitudes de recepción de todas las porciones del búfer intermedio de recepción.

   d) Realiza una petición de conexión al servidor para establecer el canal de datos.

4. El cliente espera hasta que se abre el canal de datos con el servidor. Este evento le indica que la conexión ha sido aceptada, y que la transferencia de datos que contiene el número de puerto donde enviar la petición de conexión del segundo canal está preparada.

5. A continuación, el `Solicitante` realiza las tareas descritas en el paso 5 del diagrama de la Figura 3.6:

   a) Inicializa las estructuras de datos empleadas por el canal de control.

   b) Publica la solicitud de recepción de los mensajes de control de flujo en el canal de control.

c) Envía una nueva petición de conexión al servidor dirigida al segundo puerto y aguarda la apertura del segundo canal de comunicación.

6. Una vez se ha establecido el canal de control, se inicia la transferencia de datos con el servidor.

Secuencia de pasos del servidor

1. En el extremo del servidor, la entidad `Oyente` permanece a la escucha de peticiones de conexión de los clientes a través de un puerto TCP/IP. Cuando el `Oyente` recibe una solicitud de conexión de un `Solicitante`, crea una entidad `Agente` y le traspasa la solicitud, tal y como se observa en el paso 2 del diagrama de la Figura 3.6.

2. Para atender la solicitud que recibe del `Oyente`, el `Agente` ejecuta las acciones presentadas en el paso 4 del diagrama de la Figura 3.6:

   a) Acepta la solicitud y abre una conexión TCP/IP.

   b) Selecciona un puerto para recibir la petición de conexión del canal de datos y permanece a la escucha en dicho puerto.

   c) Posteriormente, envía la dirección del puerto al `Solicitante` a través de la conexión TCP/IP.

   d) Cuando el `Agente` recibe la solicitud para establecer el canal de comunicación de datos realiza las siguientes tareas:

      1) Obtiene el identificador de la nueva conexión.
      2) Inicializa las estructuras de datos necesarias para la conexión entre el cliente y el servidor.
      3) Registra los búferes intermedios utilizados para el envío y la recepción.
      4) Publica las solicitudes de recepción de todas las porciones del búfer intermedio de recepción.
      5) Acepta la conexión y abre el canal de comunicación encargado de transportar los datos entre el cliente y el servidor.

   e) Una vez se ha establecido el canal de datos, el `Agente` abre un nuevo puerto para recibir la petición de conexión del canal de control y permanece a la escucha.

f) Después, publica la solicitud de envío de la dirección del segundo puerto al `Solicitante` a través del canal de datos.

g) Al recibir del `Solicitante` la solicitud para establecer el canal de comunicación de control, éste realiza las acciones que se enumeran en el paso 6 del diagrama de la Figura 3.6:

1) Obtiene el identificador de la nueva conexión.
2) Genera las estructuras de datos necesarias para la conexión entre el cliente y el servidor.
3) Publica la solicitud de recepción de los mensajes de control de flujo.
4) Acepta la conexión y establece el canal de comunicación encargado de transportar los mensajes de control de flujo entre los extremos.

3. Al finalizar el proceso descrito, el servidor es capaz de intercambiar datos con el cliente.

3.5. Módulo de comunicaciones basado en semántica de memoria

En esta sección se describen las características que presenta el segundo módulo de comunicaciones para rCUDA desarrollado a partir del módulo FAR. Dicho módulo, que recibe el nombre de MSR (*Memory Semantics over RDMA*), utiliza operaciones de semántica de memoria en las transferencias de datos entre el cliente y el servidor.

3.5.1. Mecanismo de transmisión

Con la intención de aumentar el rendimiento de las comunicaciones entre los clientes y los servidores de rCUDA, se han aplicado las técnicas descritas en los siguientes puntos al mecanismo de transmisión del módulo MSR.

División de los búferes intermedios de transferencia de datos

Al igual que ocurre con el módulo CSR, los búferes intermedios de envío y recepción de los datos de las peticiones se dividen en porciones.

Esto permite utilizar el mecanismo de *pipeline* de rCUDA en las transferencias del módulo, de forma que el destino puede operar con los datos de las porciones recibidas mientras el emisor ejecuta las escrituras remotas en las siguientes porciones.

Reducción del número de regiones de memoria utilizadas

Como se ha visto en la Sección 3.4, el registro incondicional de los búferes que utiliza en el módulo FAR puede tener un impacto negativo en el rendimiento de la comunicaciones. Para mitigar este sobrecoste se reduce el número de operaciones de registro de búferes del módulo de comunicaciones. En concreto, se registran los búferes intermedios utilizados para transferir la información de las peticiones y los resultados de la ejecución de las funciones de CUDA, y las porciones de los búferes intermedios de envío y recepción de datos.

Utilización de operaciones de semántica de memoria

Otra característica del mecanismo de transmisión implementado en el módulo MSR es la utilización de operaciones de semántica de memoria en las transferencias de las peticiones de ejecución y los datos asociados.

Un rasgo que distingue la semántica de memoria de la semántica de canal es que el extremo destino de la comunicación es pasivo. En concreto, el receptor registra un búfer y le pasa el control de ese búfer al remitente, que utiliza las operaciones RDMA de lectura o escritura para acceder los datos del búfer remoto. Esto plantea los siguientes problemas al utilizar esta semántica en las transferencias de rCUDA:

1. Es imprescindible que el origen de la operación RDMA disponga de la información del registro de los búferes del extremo opuesto. Para superar este primer problema, las entidades `Solicitante` y `Agente` del módulo MSR intercambian la información del registro de sus búferes utilizando las operaciones de semántica de canal después del establecimiento de la conexión.

2. El receptor no es capaz de saber si la operación RDMA se ha completado por lo que es necesario implementar en el módulo de comunicación MSR un mecanismo que le advierta de su finalización.

En este caso, el emisor efectúa una escritura y posteriormente genera un mensaje de notificación para informar al receptor que los datos ya se encuentran en destino.

3. El emisor no detectará cuándo el receptor ha leído o modificado los datos del búfer y puede ocurrir que una escritura remota del emisor destruya información que aún no ha sido procesada. Con el objetivo de evitar esto, el emisor espera hasta que el receptor le envía un mensaje de control de flujo a través del canal usado en la escritura de los datos. De esta forma, el receptor advierte al emisor que el búfer destino de los datos está disponible para una nueva escritura.

Los mensajes de control de flujo y de notificación usados por el módulo MSR se transmiten mediante operaciones de semántica de canal y tienen el campo de datos vacío.

La utilización de la semántica de canal permite establecer un mecanismo de sincronización entre los extremos basado en las WCs (*Work Completion*s) que se generan cuando las transferencias se completan.

Por otro lado, al no transportar datos, el mensaje de control de flujo o notificación no consume ancho de banda efectivo, que permanece dedicado a los datos de la aplicación.

Como resultado, el proceso de transmisión implementado en el módulo MSR es el siguiente (véase la Figura 3.7):

1. Al principio, como puede observarse en el paso 1 de la Figura 3.7, el extremo emisor publica una solicitud de recepción de un mensaje de control de flujo y el receptor emite una solicitud de recepción de un mensaje de notificación. Ambas se añaden a las ya existentes. Esto garantiza que existen solicitudes de recepción para los futuros mensajes de control de flujo. A continuación, el emisor permanece a la espera de un mensaje de control de flujo para iniciar la transferencia.

2. Cuando el receptor está preparado, envía un mensaje de control de flujo (paso 2 de la Figura 3.7).

3. Al recibirlo, como se muestra en el paso 3 de la Figura 3.7, el emisor comienza las escrituras de una porción del búfer o del búfer completo. Esto depende de si se transmiten datos desde el búfer intermedio de

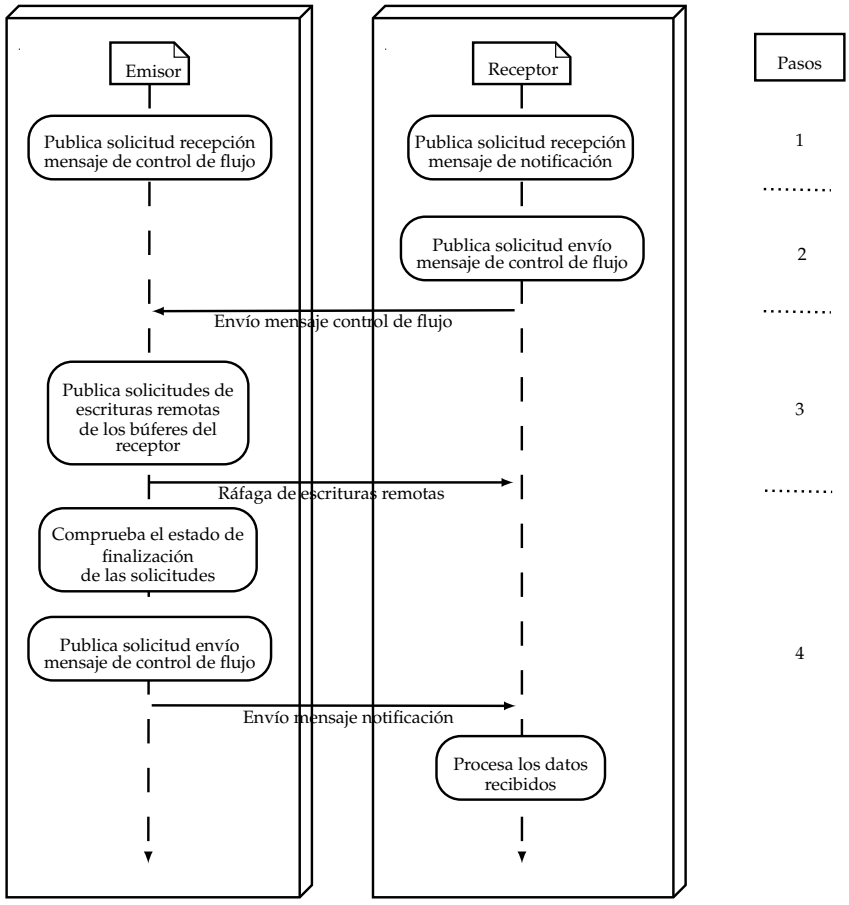


Figura 3.7: Diagrama del proceso de transmisión del módulo MSR.

envío de datos o desde cualquiera de los otros búferes del módulo de comunicación.

4. Una vez se completan las escrituras, el emisor comprueba si han habido errores, en cuyo caso repite la transferencia. Después, envía al receptor el mensaje de notificación para que procese los datos, tal y como se muestra en el paso 4 de la Figura 3.7.

5. El emisor repite los pasos 3 y 4 de la Figura 3.7 hasta que se envían todos los datos o se escribe la última porción del búfer intermedio del receptor. En caso de que se ejecute la escritura de la última porción del búfer remoto y aún queden datos por transmitir, el emisor comienza el proceso desde el paso 1 de la Figura 3.7.

Esto garantiza que el receptor ha procesado los datos recibidos y que el búfer de recepción está preparado.

Empleo de GPUDirect RDMA en las transferencias

Por último, el módulo MSR utiliza el mecanismo GPUDirect RDMA descrito en el Apartado 2.6.3. De esta forma se consigue reducir el tiempo necesario para copiar los datos de los búferes intermedios del módulo de comunicación a la zona definitiva de la memoria de la GPU.

3.5.2. Creación del doble canal de comunicación

El desarrollo MSR reutiliza el mecanismo del doble canal de comunicación que se describe en el Apartado 3.2.1 para en el módulo FAR, donde cada canal transporta los datos y los mensajes de notificación en una única dirección, mientras que los mensajes de control de flujo discurren por ese canal en sentido contrario a los anteriores (Figura 3.8).

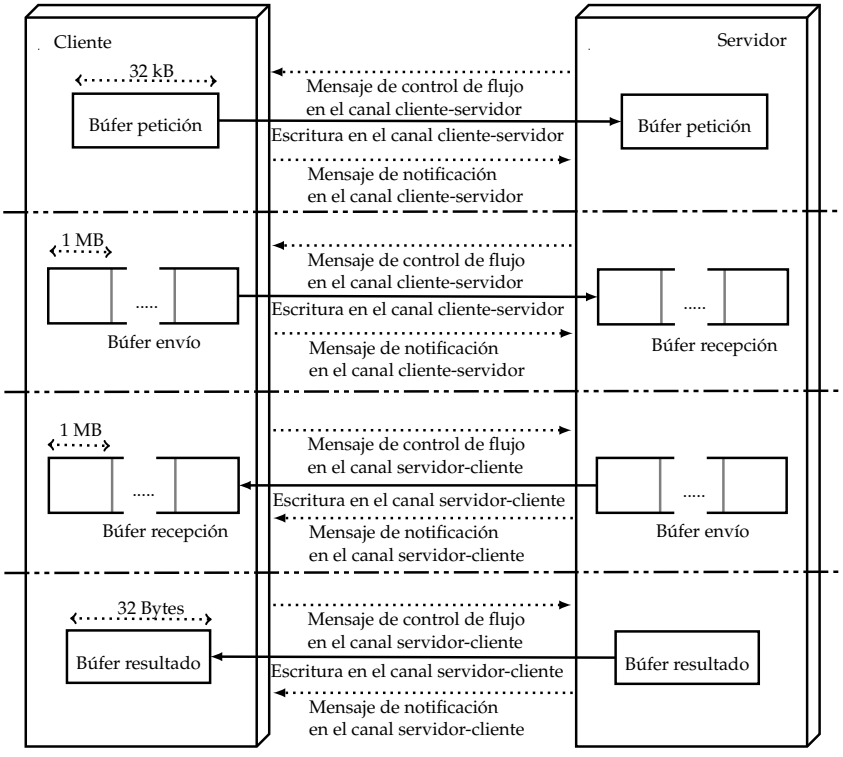


Figura 3.8: Diagrama de búferes y canales utilizados en el módulo MSR.

No obstante, para que el servidor de rCUDA acepte múltiples conexiones simultáneas, es necesario redefinir el protocolo de creación del doble canal. Por eso, del mismo modo que ocurre en el módulo CSR, el módulo MSR utiliza sockets POSIX durante el establecimiento de la conexión.

Al comienzo, el servidor rCUDA dispone de un socket que escucha las peticiones de conexión de los clientes. Cuando se recibe una petición, se crea un subproceso al que se le transfiere el control de la nueva conexión TCP/IP. El subproceso abre un nuevo puerto para recibir la solicitud de conexión RDMA y le envía al cliente su dirección. Una vez recibida la información del puerto, el cliente envía la petición de la conexión a ese puerto. En ese momento, el subproceso del servidor crea las estructuras de datos de la conexión RDMA.

Las Figuras 3.9 y 3.10 muestran los pasos que sigue el módulo MSR para el establecimiento de la conexión y la creación del doble canal RDMA. A continuación se proporciona una descripción de cada uno de ellos.

Secuencia de pasos del cliente

1. En primer lugar el cliente realiza una petición de conexión TCP/IP al servidor y permanece a la espera hasta que se abre el canal de comunicación con el servidor.

2. Cuando se recibe el número de puerto donde enviar la petición de conexión del canal servidor-cliente, la entidad `Solicitante` del cliente ejecuta las acciones que se enumeran en el paso 3 del diagrama de la Figura 3.9:

   a) Genera las estructuras de datos necesarias para la comunicación.

   b) Publica una solicitud de recepción para obtener el número de puerto de la petición de conexión del canal cliente-servidor.

   c) Realiza una petición de conexión al servidor para establecer el canal servidor-cliente.

3. Como se expone en el paso 5 del diagrama de la Figura 3.9, una vez recibe el segundo número de puerto, el `Solicitante` lleva a cabo las tareas que se enumeran a continuación:

   a) Crea las estructuras de datos usadas por el canal cliente-servidor.

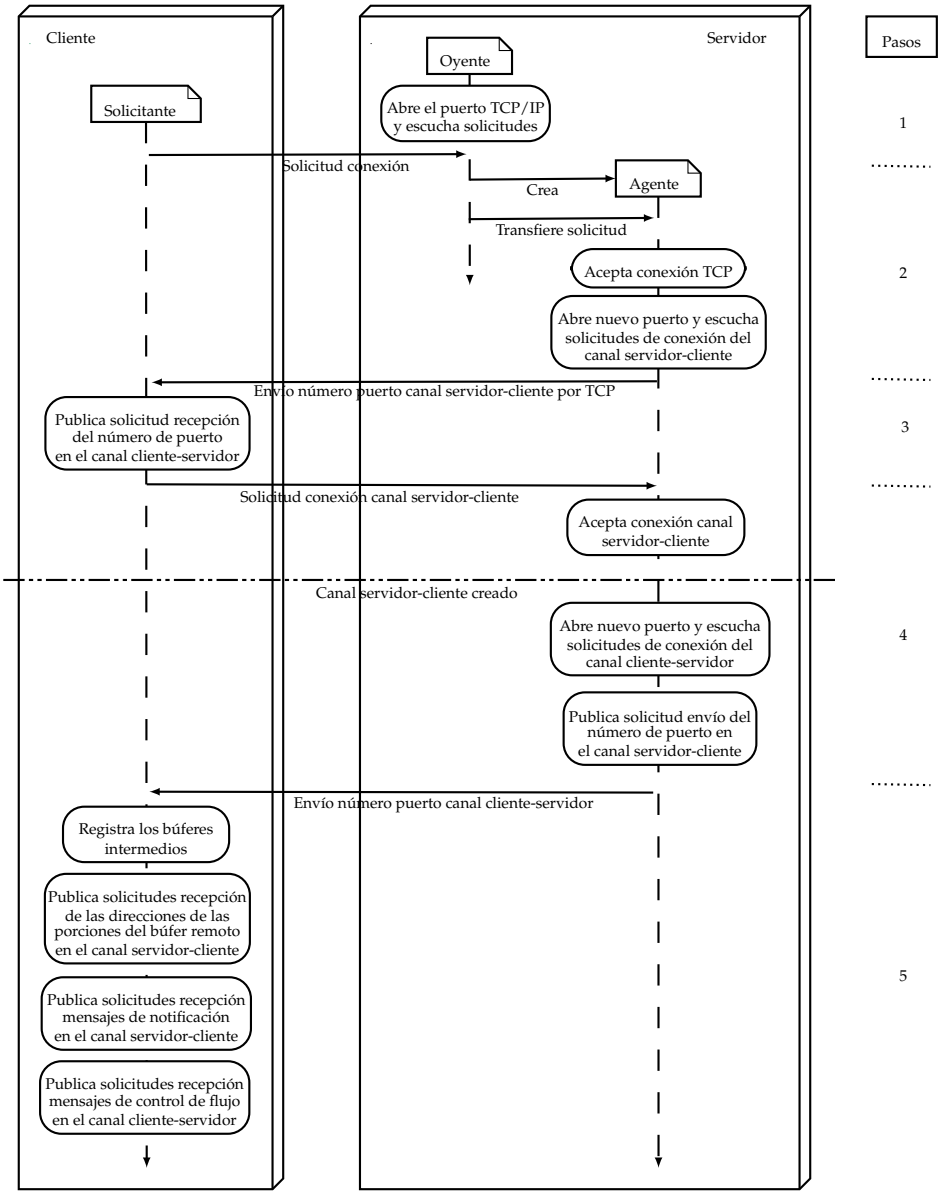


Figura 3.9: Diagrama del establecimiento de conexión del módulo MSR (pasos 1-5).

b) Registra los búferes intermedios empleados en las escrituras remotas.

c) Emite la solicitud de recepción de las direcciones de memoria de los búferes del servidor.

d) Publica las solicitudes de recepción de los mensajes de control

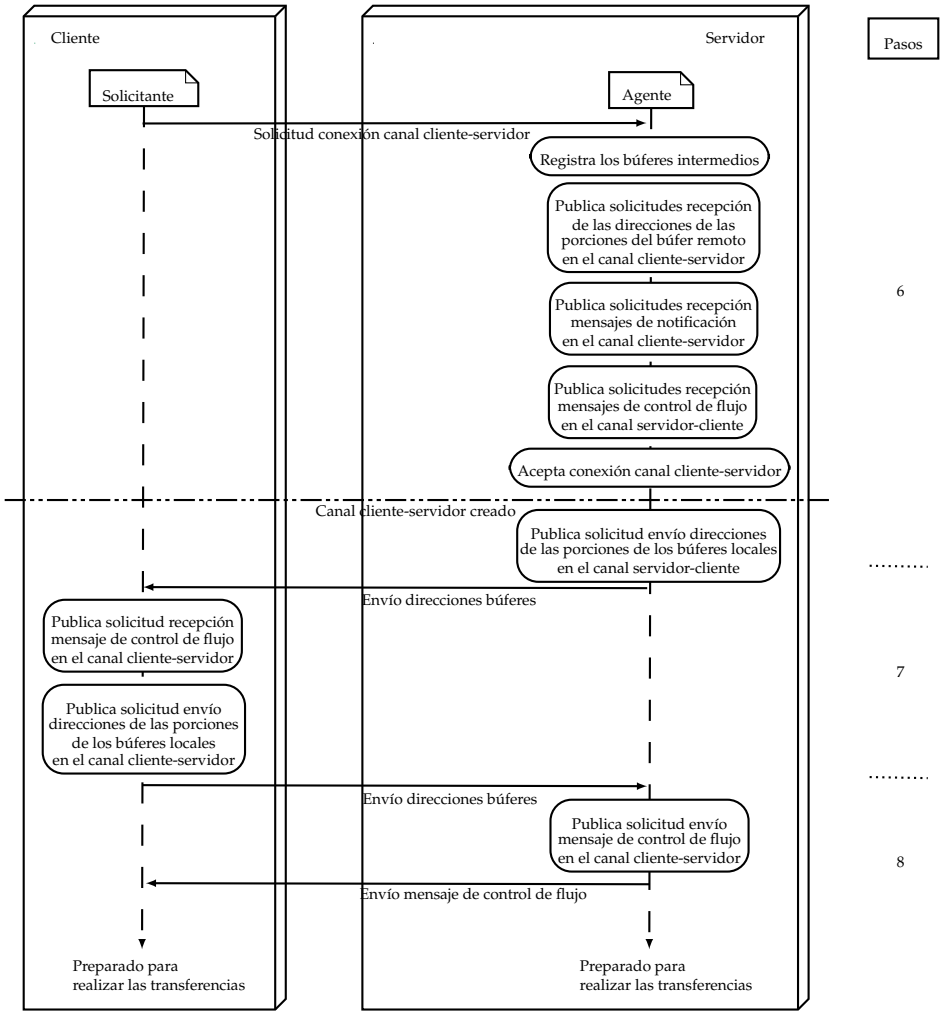


Figura 3.10: Diagrama del establecimiento de conexión del módulo MSR (pasos 6-8).

de flujo en el canal cliente-servidor y las peticiones de recepción de los mensajes de notificación en el canal servidor-cliente.

e) Realiza una nueva petición de conexión al servidor dirigida al segundo puerto, y aguarda a que el segundo canal de comunicación, el canal cliente-servidor, se encuentre operativo.

4. Una vez se ha abierto el canal cliente-servidor y se han recibido las direcciones de memoria de los búferes remotos, el `Solicitante` publica la solicitud de envío de las direcciones de memoria de los búferes locales.

5. Para finalizar, el `Solicitante` emite una solicitud de recepción de un mensaje de control de flujo en el canal cliente-servidor. La recepción de este mensaje indica al cliente que puede comenzar las transferencias de datos con el servidor utilizando operaciones de escritura remota.

Secuencia de pasos del servidor

1. En el diagrama de la Figura 3.9 puede observarse como la entidad `Oyente` del servidor aguarda la recepción de peticiones de conexión de los clientes a través de un puerto TCP/IP. Tras la llegada de una solicitud, el `Oyente` crea una entidad `Agente` y le traspasa la solicitud.

2. La entidad `Agente` se encarga de realizar el siguiente trabajo:

   a) Como se muestra en el paso 2 del diagrama de la Figura 3.9, establece una conexión TCP/IP con el `Solicitante`.

   b) Además, reserva un puerto para recibir la petición de conexión del canal de datos y permanece a la escucha.

   c) A continuación, envía la dirección del puerto al cliente a través de la conexión TCP/IP.

   d) Después de recibir la solicitud de conexión, el `Agente` realiza las siguientes tareas:

      1) Obtiene el identificador de conexión.
      2) Crea las estructuras de datos necesarias para la conexión entre el servidor y el cliente.
      3) Acepta la conexión y abre el canal de comunicación utilizado para enviar los datos al cliente (canal servidor-cliente).

   e) El `Agente` prosigue con el proceso abriendo un nuevo puerto para atender la petición de conexión del segundo canal de comunicación.

   f) Posteriormente, publica la solicitud de envío de la dirección de ese puerto al `Solicitante` a través del canal servidor-cliente.

g) Cuando se recibe la segunda solicitud de conexión, como se representa en el paso 6 del diagrama de la Figura 3.10, el `Agente` lleva a cabo estas acciones:

1) Obtiene el identificador de la nueva conexión.
2) Inicializa las estructuras de datos necesarias para la conexión entre el cliente y el servidor.
3) Registra los búferes intermedios utilizados para realizar las escrituras remotas.
4) Publica las solicitudes de recepción de los mensajes de control de flujo en el canal servidor-cliente y las peticiones de recepción de los mensajes de notificación en el canal cliente-servidor.
5) Publica la solicitud de recepción de las direcciones de memoria de los búferes del cliente.
6) Acepta la conexión y establece el canal de comunicación cliente-servidor, que se encarga de transportar los datos desde el cliente hasta el servidor.
7) Publica la solicitud de envío de las direcciones de memoria de los búferes locales y espera la recepción de las direcciones de los búferes remotos.

h) Para finalizar, una vez se han procesado las direcciones de memoria de los búferes remotos, el `Agente` publica una solicitud de envío de un mensaje de control de flujo en el canal cliente-servidor. Este mensaje informa al cliente de que las estructuras de datos y los búferes del servidor se encuentran preparados para recibir las escrituras remotas.

3.6. Ajustes avanzados de los módulos para redes RDMA

Los módulos CSR y MSR disponen de varios parámetros que les permiten adaptarse a los requerimientos de cada escenario en los que se ejecuta la solución rCUDA. Concretamente, es posible deshabilitar el mecanismo GPUDirect RDMA, establecer la utilización de búferes intermedios

dedicados para las peticiones y las respuestas de las funciones de CUDA, indicar el método de consulta de las colas de finalización, definir el intervalo de tiempo entre las consultas y determinar el número de porciones de los búferes intermedios de comunicación.

En los dos apartados siguientes se describen los dos parámetros principales de los módulos CSR y MSR, y se analiza el impacto que pueden tener sobre la solución de virtualización de rCUDA. Para concluir, se proponen los valores óptimos de cada uno de ellos para el presente trabajo.

3.6.1. Método de consulta de finalización de operaciones

Como se indicó en el Apartado 2.3.1, al completar la operación asociada a un WQE (*Work Queue Element*) se genera un CQE (*Completion Queue Element*), que se inserta en la cola de finalización o CQ. Así pues, es necesario consultar periódicamente la CQ para saber cuál ha sido el estado de finalización de las solicitudes de trabajo o WRs publicadas.

Dentro de los módulos CSR y MSR se implementan dos métodos de consulta de la CQ:

Consulta Activa o BP (*Busy Polling*). Este método consiste en comprobar el estado de la CQ a intervalos de tiempo regulares. Esto hace que el sistema consuma tiempo de ejecución en realizar la tarea de consulta de forma explícita.

Espera de un Evento o WFE (*Wait For Event*). Mediante el uso de llamadas del SO se interrumpe la ejecución del proceso de transmisión. El bloqueo finaliza cuando el sistema informa de que se ha producido el evento asociado al cambio de estado de la CQ.

Para examinar su impacto se ha realizado un análisis comparativo del rendimiento y del consumo energético de las ejecuciones de la aplicación `bandwidthTest`. En particular, se han probado ambos métodos de consulta sobre un escenario compuesto por las infraestructuras descritas en el Apartado 3.2.3, donde el cliente y el servidor de rCUDA se comunican mediante una red IB.

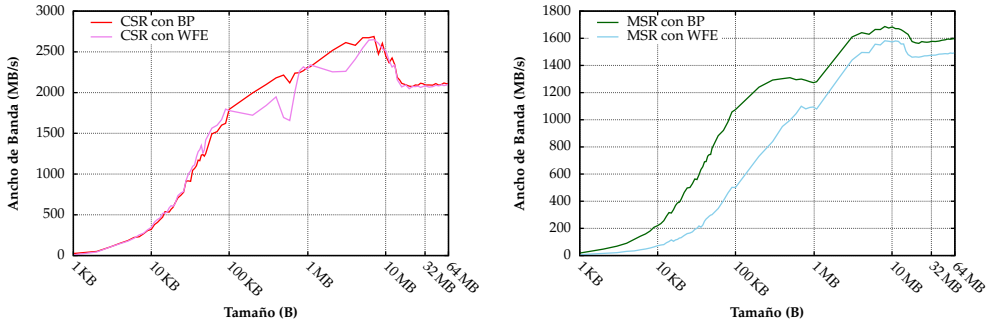
El rendimiento se ha obtenido a través de la mediciones facilitadas por `bandwidthTest` en cada una de las ejecuciones, que consisten en la

media del ancho de banda de 10 transferencias seguidas para un conjunto de datos con dimensiones comprendidas desde 1 KB hasta 64 MB.

Los datos de consumo energético han sido proporcionados por un medidor Watts Up? PRO [100], conectado a la linea eléctrica de la fuente de alimentación o PSU (*Power Supply Unit*), que devuelve la potencia instantánea con una precisión de ±1,5 % y una frecuencia de una muestra por segundo. Las medidas han sido almacenadas en un servidor separado con el fin de que este proceso no interfiera en la exactitud de los resultados.

El consumo eléctrico se ha calculado como la diferencia entre la energía consumida por las máquinas en estado de reposo y la energía consumida al ejecutar la aplicación.

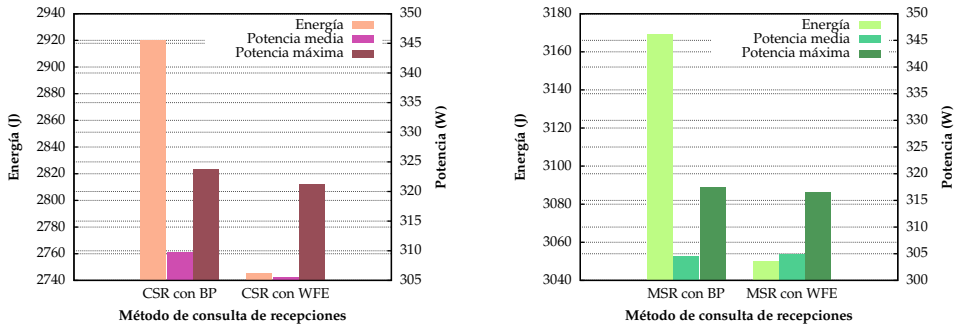
Los resultados de aplicar cada uno de los métodos de consulta sobre los módulos CSR y MSR se representan en las Figuras 3.11 y 3.12.



(a) CSR **(b)** MSR

Figura 3.11: Rendimiento del módulos CSR y MSR en función del método de consulta de la cola de finalización de solicitudes.

El primer aspecto que resalta en la Figura 3.11 es el elevado rendimiento obtenido al utilizar el método BP. Concretamente, en CSR, el incremento está entre un 0,2 % y un 30,1 %, con un promedio del 4,87 % y donde la mayor divergencia se encuentra en los tamaños cercanos a 512 KB. En el caso de MSR se alcanza una diferencia máxima del 114,4 % para los envíos de 100 KB, que se reduce hasta 5,7 % con tamaños superiores a 1 MB. Además, el rendimiento medio para BP es 2,07 × veces mayor al obtenido con WFE. Esto se debe principalmente a que el método BP detecta la finalización de una operación de comunicación antes que WFE, por lo que este primer método de consulta reduce el tiempo de ejecución.



(a) CSR **(b)** MSR

Figura 3.12: Requisitos energéticos de los módulos CSR y MSR en función del método de consulta de la cola de finalización de solicitudes.

Sin embargo, cuando se compara el consumo energético de ambos métodos (Figura 3.12), se advierte que el método BP requiere más recursos que WFE. Un examen detallado de la Figura 3.12 muestra que, en caso del módulo CSR, las ejecuciones con BP consumen un 6,3 % más que las que emplean WFE, mientras que para el módulo MSR, la diferencia es el 3,9 %. El mayor consumo de las ejecuciones con BP es consecuencia de que este método realiza la consulta de una forma explícita y utiliza más recursos del sistema para llevarla a cabo.

Así pues, con la intención de maximizar el rendimiento de las comunicaciones y, por tanto, reducir el tiempo de ejecución de la aplicación, lo que normalmente se traduce en un menor consumo energético [7, 70, 101, 102], en el presente trabajo, los módulos CSR y MSR utilizarán el método BP para consultar las colas de finalización.

3.6.2. Número de porciones de los búferes intermedios

El segundo parámetro de los módulos de comunicaciones CSR y MSR a tener en cuenta es el número de porciones que forman los búferes intermedios de envío y de recepción.

En [91] se indica que el tamaño óptimo para los datos transportados en las transferencias de rCUDA es de 1 MB puesto que ese valor permite obtener el rendimiento máximo en las copias entre la memoria principal del servidor y el acelerador gráfico utilizando la técnica de *pipeline* expuesta en el Apartado 2.6.3.

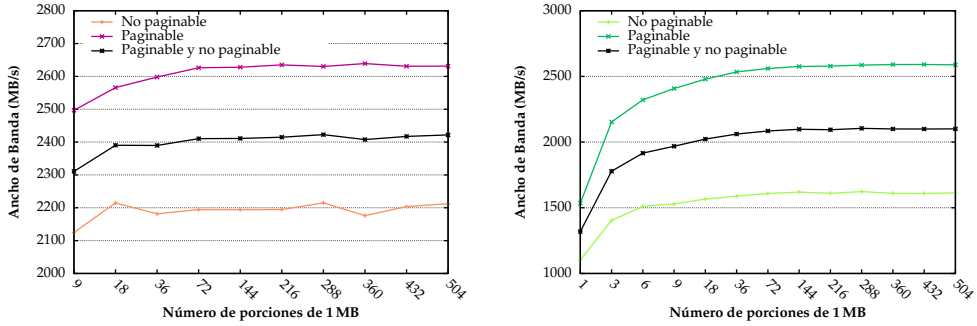
De acuerdo con lo anterior, los búferes intermedios se deben dividir en porciones de 1 MB, de manera que las operaciones de comunicación solo se ejecutan sobre las porciones, y no sobre el búfer completo como una unidad. Esto condiciona la longitud de los datos que se pueden enviar o recibir en una WR, ya que nunca se puede superar el tamaño de la porción.

Por otro lado, según el funcionamiento de los módulos CSR y MSR descrito en las Secciones 3.4 y 3.5, cuando se transmite la última porción de una ráfaga o un búfer, el emisor espera un mensaje de control del receptor que le indique que puede continuar la transmisión. En consecuencia, si se dividen los búferes en un número pequeño de porciones, aumentará el porcentaje de mensajes de control con respecto a las transferencias de datos, debido a que cada vez que se envía la última porción del búfer es necesario el envío de un mensaje de control de flujo para continuar. Esto supone una reducción del ancho de banda dedicado al transporte de datos de la aplicación.

Otro factor importante en la elección del número de porciones es la memoria del adaptador de canal. Al realizar una transferencia de datos se asocia una porción de un búfer con una solicitud de trabajo o WR, de manera que cada porción debe estar registrada en la tabla almacenada en la memoria del adaptador de canal. Por tanto, si se emplea un elevado número de porciones, existe la posibilidad de agotar la memoria del adaptador. Así, por ejemplo, el adaptador de canal para las redes iWARP utilizado en el presente trabajo no dispone de capacidad suficiente para que módulo MSR divida los búferes intermedios en más de 72 porciones.

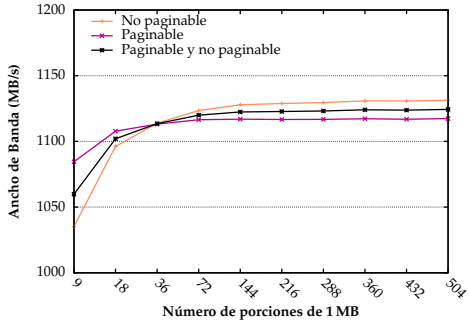
Además, se debe tener en cuenta la limitación impuesta por el mecanismo de transmisión implementado por el módulo CSR que ha sido descrita en el Apartado 3.4.2. Concretamente, el módulo CSR requiere que los búferes intermedios dispongan de al menos 9 porciones y que dicho número sea múltiplo de 3.

Para evaluar la incidencia que tiene la cantidad de porciones de los búferes intermedios en el rendimiento de los módulos de comunicación de rCUDA, se ha ejecutado la aplicación `bandwidthTest` sobre las configuraciones descritas en el Apartado 3.2.3, en las que cliente y servidor se comunican a través de una infraestructura IB, RoCE y iWARP. La aplicación `bandwidthTest` ha sido configurada para proporcionar el ancho de banda tanto para memoria paginable como para memoria no paginable.

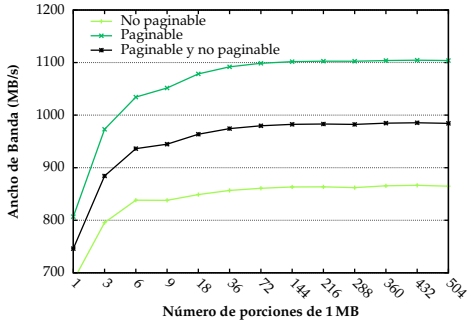


(a) CSR sobre InfiniBand

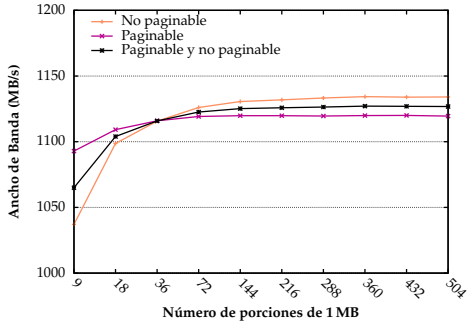
(b) MSR sobre InfiniBand



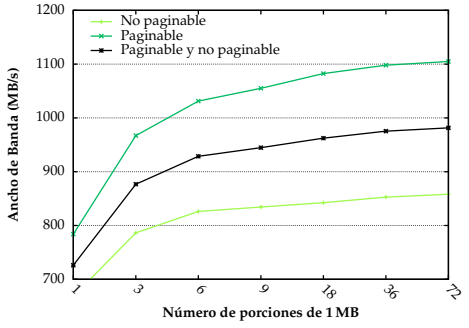
(c) CSR sobre RoCE



(d) MSR sobre RoCE



(e) CSR sobre iWARP



(f) MSR sobre iWARP

Figura 3.13: Rendimiento de los módulos CSR y MSR en función del número de porciones de 1 MB de los búferes intermedios.

Esto ha permitido efectuar un estudio del rendimiento medio de los módulos sobre cada una de las tecnologías RDMA en función del número de porciones de los búferes intermedios. En la Figura 3.13 se muestra en detalle el ancho de banda obtenido por `bandwidthTest` al realizar 10 transferencias consecutivas de 32 MB utilizando diferentes números de porciones para los búferes intermedios de envío y de recepción.

Porciones	Ancho de banda (MB/s)					
	CSR			MSR		
	IB	RoCE	iWARP	IB	RoCE	iWARP
1	–	–	–	1.318,92	745,97	726,03
3	–	–	–	1.778,48	884,20	876,70
6	–	–	–	1.915,82	936,22	928,67
9	2.310,75	1.059,80	1.065,00	1.968,55	944,75	944,67
18	2.390,23	1.102,00	1.103,95	2.022,45	963,65	962,40
36	2.389,80	1.113,50	1.115,82	2.061,22	974,33	975,38
72	2.410,40	1.119,97	1.122,62	2.084,47	979,77	981,48
144	2.411,05	1.122,42	1.125,20	2.097,40	982,52	–
216	2.414,82	1.122,78	1.125,85	2.093,97	983,15	–
288	2.422,62	1.124,97	1.126,42	2.104,53	982,30	–
360	2.407,60	1.124,05	1.127,12	2.100,00	984,73	–
432	2.417,23	1.123,83	1.126,97	2.099,57	985,48	–
504	2.423,85	1.124,35	1.126,80	2.100,60	984,32	–

Tabla 3.1: Rendimiento de los módulos CSR y MSR según el número de porciones de 1 MB de los búferes intermedios.

		Módulo CSR	Módulo MSR
Tecnología RDMA	InfiniBand	288	288
	RoCE	144	144
	iWARP	72	72

Tabla 3.2: Número óptimo de porciones de 1 MB para los búferes intermedios de los módulos CSR y MSR.

Con el objetivo de escoger un valor para la cantidad de porciones de los búferes intermedios que maximice el rendimiento de los módulos de comunicación en cada una de las infraestructuras de red, se ha construido la Tabla 3.1. En ella se disponen los rendimientos medios de los módulos (calculados a partir del ancho de banda de las transferencias con memoria paginable y memoria no paginable) en función de las infraestructuras de red y del número de porciones.

Al analizar en detalle los valores de la Tabla 3.1 y los gráficos de la Figura 3.13, se observa como en todos los casos existe un punto en el que la función asociada al rendimiento medio alcanza un estado estacionario.

Estos valores, que se recogen en la Tabla 3.2, señalan el número óptimo de porciones que debe ser utilizado por los módulos para cada una de las tecnologías RDMA, ya que maximiza el rendimiento y minimiza la cantidad de memoria utilizada. En consecuencia, estos valores serán empleados en las configuraciones de los módulos CSR y MSR en el resto del presente trabajo.

3.7. Análisis de viabilidad de los módulos para redes RDMA

En esta sección se examina la posibilidad de emplear los desarrollos presentados en las Secciones 3.4 y 3.5 como soluciones productivas de comunicación para rCUDA. Así pues, se debe demostrar que ambos módulos proporcionan a rCUDA la funcionalidad necesaria para acceder eficientemente a las GPUs remotas a través de redes RDMA a las aplicaciones CUDA. Hasta el momento se ha empleado el módulo TCP cuando las redes RDMA implementaban el modelo TCP/IP y no eran soportadas por el módulo IB. Sin embargo, esto reduce el rendimiento de la virtualización debido a la sobrecarga que introduce la pila de protocolos TCP/IP [91]. Los módulos CSR y MSR han sido diseñados con la intención de aprovechar los mecanismos que ofrecen las tecnologías RDMA, maximizando el rendimiento de las transferencias y mejorando el rendimiento obtenido por el módulo TCP.

Con el propósito de confirmar estas ventajas, se ha realizado un análisis comparativo del ancho de banda de las transferencias entre una CPU y una GPU virtualizada con rCUDA. Para ello se ha ejecutado nuevamente la aplicación `bandwidthTest` sobre la infraestructura descrita en el Apartado 3.2.3 y se ha evaluado el rendimiento para cada uno de los módulos de comunicación sobre las tres tecnologías RDMA (IB, RoCE y iWARP). Para evitar el ruido, las pruebas se han repetido 3 veces. En las Figuras 3.14, 3.15 y 3.16 se han representado los resultados obtenidos en la mejor de esas 3 ejecuciones. La desviación estándar relativa media observada en CSR ha sido del 13,79 % mientras que para MSR ha alcanzado el 4,89 %, encontrándose la desviación relativa máxima en ambos módulos en las transferencias menores a 100 KB.

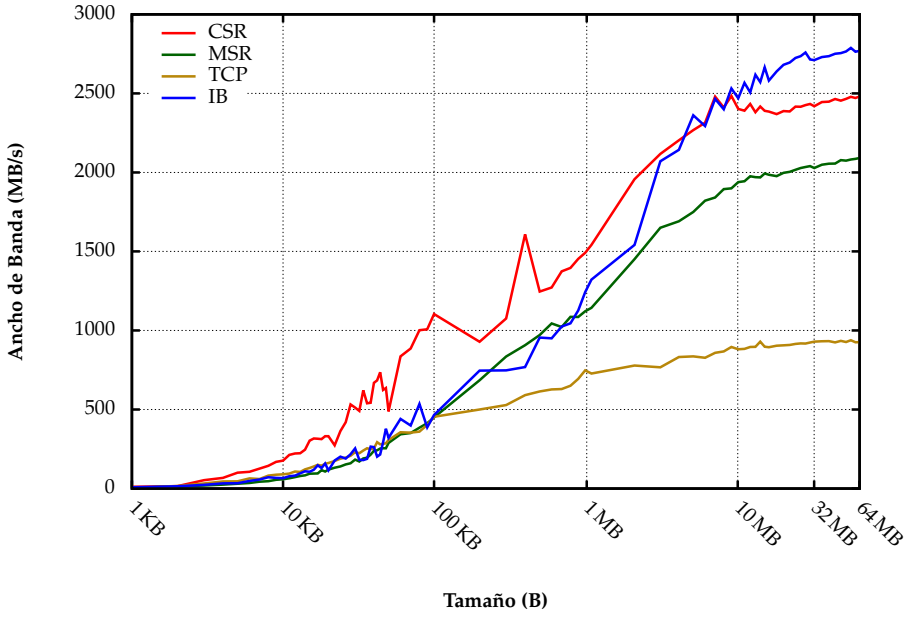


Figura 3.14: Evaluación del rendimiento de los módulos CSR y MSR sobre IB.

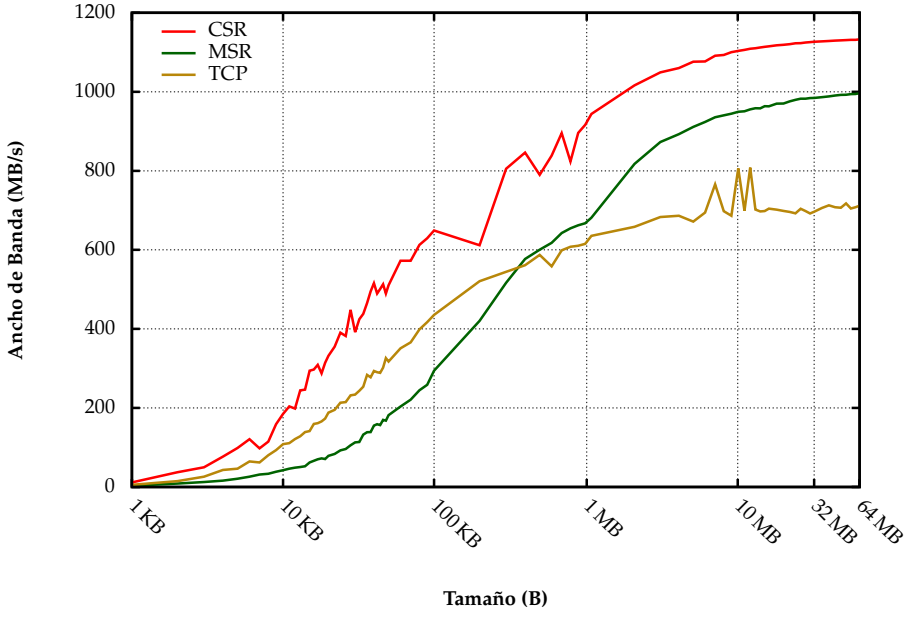


Figura 3.15: Evaluación del rendimiento de los módulos CSR y MSR sobre iWARP.

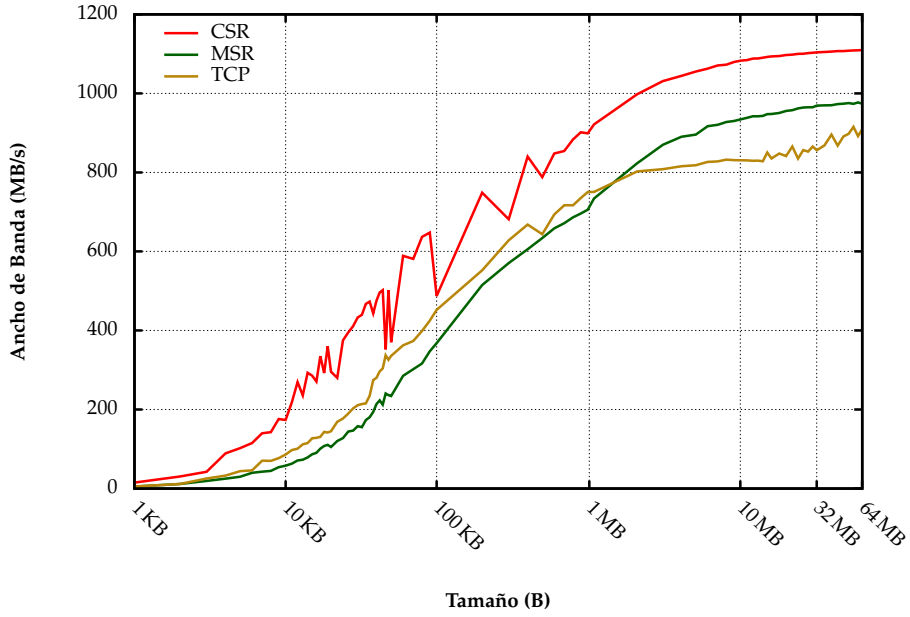


Figura 3.16: Evaluación del rendimiento de los módulos CSR y MSR sobre RoCE.

El aspecto más destacable del análisis de las Figuras 3.14, 3.15 y 3.16 es que el módulo CSR supera el rendimiento del módulo TCP sobre cualquier infraestructura de red RDMA. Incluso en la Figura 3.14 puede verse como el módulo CSR obtiene mejores resultados que el módulo IB para transferencias de datos de tamaños inferiores a 10 MB. En concreto, el incremento llega hasta un 122 % en el caso de tamaños entre 10 KB y 512 KB para la infraestructura de red IB.

Por otro lado, en la Figura 3.14 se puede comprobar como el módulo MSR puede mejorar el rendimiento del módulo TCP hasta un 120,6 % cuando se envían mensajes de más 100 KB. Sin embargo, este rendimiento no llega a superar al del módulo IB, y en el peor de los casos obtiene un 79 % de su ancho de banda.

En lo referente al rendimiento del módulo MSR en las configuraciones con redes iWARP y RoCE , como puede observarse en las Figuras 3.15 y 3.16, éste mejora hasta un 47,9 % y un 11,4 %, respectivamente, el ancho de banda presentado por el módulo TCP en las transmisiones de datos de tamaños superiores a 1 MB. Sin embargo, para tamaños menores, el ancho de banda del módulo TCP sigue por encima del obtenido por módulo MSR.

Esta circunstancia se debe a que, para transmisiones con un tamaño pequeño, inferior a 100 KB en IB y a 1 MB en iWARP y RoCE, la penalización del envío de un mensaje de control y de un mensaje de notificación por cada escritura remota que efectúa el módulo MSR es muy elevada. Como se describió en el Apartado 3.5.1, en estos casos a la transferencia con los datos se le suman dos transferencias adicionales: una para el mensaje de control de flujo y otra para el mensaje de notificación. Además, a esto se añade que los dispositivos de conexión RoCE y iWARP disponen de mecanismos, como por ejemplo la tecnología TOE (*TCP Offload Engine*), que minimizan la sobrecarga que introducen los protocolos TCP/IP, y que funcionan especialmente bien para tamaños pequeños.

En definitiva, a pesar de esto, los resultados expuestos prueban que los módulos CSR y MSR pueden emplearse como soluciones productivas de comunicación de rCUDA para cualquier infraestructura de red con soporte para RDMA. Así pues, de ahora en adelante, se descarta la utilización del módulo TCP para las redes RDMA en el presente trabajo, ya que las soluciones CSR y MSR, aprovechando las ventajas que ofrecen las tecnologías RDMA, permiten realizar transferencias con rendimientos superiores al módulo TCP.

3.8. Resumen y conclusiones

En este capítulo se han descrito las ampliaciones de rCUDA desarrolladas dentro del presente trabajo de investigación con la intención de explotar los beneficios que proporcionan las infraestructuras de red que soportan RDMA descritas en la Sección 2.4.

Concretamente, al comienzo del capítulo se ha detallado la arquitectura compartida por los módulos de comunicación diseñados para rCUDA. A continuación, se han introducido los mecanismos utilizados por la aproximación inicial del módulo de comunicaciones para aumentar el rendimiento de las transferencias. En este sentido, el doble canal de comunicación y la agrupación de transferencias son las tecnologías más destacables desarrolladas como aportación del presente trabajo e integradas en las soluciones finales. Posteriormente, se ha evaluado su rendimiento sobre cada una de las tecnologías RDMA frente a los módulos de rCUDA existentes,

demostrándose que, a pesar de contar con ciertas carencias, es un buen punto de partida para el desarrollo de las soluciones completas.

Para continuar, se ha presentado la primera de las soluciones completas desarrolladas en esta investigación, denominada CSR. Como se ha mostrado, el módulo CSR se basa en la semántica de canal para realizar las transferencias y utiliza un doble canal dedicado. De forma que los datos circulan por un canal de comunicación y los mensajes de control de flujo por el otro. Además, se ha examinado el proceso de establecimiento de este doble canal entre el cliente y el servidor, y se han descrito los mecanismos empleados para maximizar el rendimiento de las transferencias. En particular, a la agrupación de transferencias, que ya había sido introducida en la aproximación inicial, se ha sumado el empleo de búferes intermedios, la notificación de recepciones con antelación, la comprobación tardía de errores, y el mecanismo de GPUDirect RDMA.

Además, el capítulo ha introducido la segunda solución completa aportada por este trabajo, el módulo MSR. Este módulo se basa en la semántica de memoria y, como se ha podido observar, dispone de un complejo proceso de establecimiento de un doble canal en el que se intercambian las direcciones de los búferes destinados a las escrituras remotas. Igualmente, se ha examinado el mecanismo de transmisión implementado por el módulo, que integra la escrituras remotas junto con los mensajes de control y de notificación para realizar las transferencias.

Posteriormente, se ha proporcionado una descripción detallada de los parámetros que permiten ajustar el rendimiento y el consumo de los módulos de comunicaciones CSR y MSR. Asimismo, se han expuesto las pruebas realizadas para la determinación de los valores óptimos de estos parámetros.

Por último, se ha realizado un análisis comparativo del rendimiento de los módulos de comunicación de rCUDA expuestos en el presente capítulo y los existentes sobre cada una de las tecnologías RDMA. Los resultados han demostrado la viabilidad de la soluciones desarrolladas. Aún más, para transferencias con tamaños de datos pequeños, el módulo CSR ha proporcionado más del doble de ancho de banda que los módulos existentes que utilizan comunicaciones TCP/IP o IB.

CAPÍTULO 4

Evaluación experimental de rCUDA en combinación con las tecnologías HPC actuales y de bajo consumo

En este capítulo se examina la posibilidad de utilizar rCUDA en un entorno heterogéneo, con clientes y servidores ejecutándose en nodos con

distintas prestaciones. Concretamente, este capítulo presenta las siguientes aportaciones:

- Un estudio de las posibles configuraciones con las tecnologías representativas del hardware utilizado hoy en día en HPC y las alternativas actuales de bajo consumo que permiten la virtualización de GPUs. En particular, las configuraciones hardware evaluadas incluyen una variedad de escenarios, con los clientes y los servidores rCUDA ejecutándose en nodos interconectados con infraestructuras de red 1GbE o RDMA, y formados por distintos tipos de procesadores multinúcleo de propósito general y varios tipos de GPUs, respectivamente.

- Una evaluación del rendimiento de las transferencias entre la CPU y la GPU de las distintas configuraciones con rCUDA representativas de las tecnologías HPC actuales.

- Un análisis experimental que identifica el potencial y los posibles sobrecostes que afectan al rendimiento y al consumo energético de las ejecuciones de aplicaciones científicas paralelas aceleradas con GPUs en las configuraciones hardware evaluadas. Para ello se han elegido 4 aplicaciones complejas que utilizan CUDA y que poseen especial relevancia, ya sea por su utilización en la evaluación del rendimiento de los HPCCs de la lista TOP500 [18], o por su papel en la investigación científica actual [80]. En concreto, se han considerado el benchmark HPL-Fermi, y las aplicaciones de producción LAMMPS, CUDASW++ y GPU-BLAST.

El resto del capítulo se estructura del siguiente modo: en la Sección 4.1 se describen detalladamente las distintas configuraciones del entorno de experimentación, así como las plataformas y las infraestructuras de red que las componen. Posteriormente, en la Sección 4.2 se realiza la caracterización de las transferencias entre la CPU y la GPU de cada una de las configuraciones hardware del entorno de experimentación. A continuación, en la Sección 4.3 se exploran los posibles beneficios y sobrecostes de la virtualización de GPUs para las aplicaciones paralelas mediante un análisis comparativo del rendimiento y del consumo energético de la ejecución de las cuatro aplicaciones seleccionadas sobre las distintas configuraciones.

Para finalizar, en la Sección 4.9 se presentan las conclusiones de este capítulo y un resumen del trabajo realizado.

4.1. Entorno de experimentación

Esta sección describe el entorno experimental utilizado para efectuar las pruebas. En primer lugar se presentan las diversas plataformas hardware; a continuación se enumeran los dispositivos de interconexión que forman parte de las infraestructuras de interconexión utilizadas en la experimentación, y para cerrar la sección, se introducen las configuraciones empleadas en los escenarios de prueba para obtener los resultados experimentales recogidos en en el resto del capítulo.

4.1.1. Plataformas

Para efectuar la experimentación se ha reunido un conjunto heterogéneo de plataformas representativas del hardware utilizado hoy en día en HPC. Concretamente, el grupo está formado por 7 máquinas equipadas con arquitecturas emergentes de bajo consumo empleadas en el diseño de HPCC actuales [24, 26, 30]; y por 3 equipos con procesadores de propósito general Intel Xeon, presentes en el 85,1 % de los sistemas de la lista TOP500 (a fecha de junio de 2015) [18].

A continuación se proporcionan las características más significativas de estos equipos cuyo resumen se ofrece en la Tabla 4.1.

Plataformas con tecnologías de bajo consumo

Carma1 y **Carma2** son dos máquinas que están equipadas con sendos procesadores NVIDIA Tegra 3 ARM Cortex A9 quad-core a 1,4 GHz, y disponen de 2 GB de memoria RAM DDR3 cada una.

Cada equipo cuenta con una GPU NVIDIA Quadro 1000M "Fermi" de 96 núcleos CUDA y con 2 GB de memoria RAM DDR3 que se comunica con el procesador a través de un bus PCIe 1.0 x4. Ambos componentes, procesador y GPU, están montados en una placa base de diseño específico [103] que cuenta con un adaptador de red Gigabit Ethernet Intel 82574L y que carece de puertos de expansión

PCIe o PCI. La potencia de diseño térmico o TDP (*Thermal Design Power*) del SoC es de 40 W.

El sistema dispone el compilador GNU gcc/g++ versión 4.6.3, el software OpenMPI 1.8.3, ATLAS 3.8.4, FFTW 3.2.1 y CUDA 6.0 sobre un SO Linux Ubuntu 12.04 con el kernel 3.1.10.

Para evitar las posibles interferencias del sistema vSMP en las pruebas se desactiva el "núcleo acompañante" de la CPU y se configuran los 4 núcleos principales a la máxima frecuencia disponible.

Kayla1 es un procesador multinúcleo similar a Carma1 y Carma2 pero con la diferencia de que no incluye ninguna GPU integrada en la placa base. En su lugar dispone de un puerto de expansión PCIe x16 con electrónica PCIe 1.0 x4 al que se ha conectado una GPU NVIDIA GeForce GTX480 "Fermi" con 448 núcleos CUDA, 1.280 MB de RAM DDR3/GDDR5. Según las especificaciones descritas por los fabricantes, la TDP total del sistema es de 260 W.

Por otro lado, la política de gestión de los núcleos y el resto de componentes del sistema son los mismos que se han descrito para Carma1 y Carma2.

Jetson1 y **Jetson2** son la evolución de Carma1 y Carma2. Presentan una arquitectura de 32 bits basada en un chip NVIDIA Tegra K1 que incluye un procesador ARM Cortex A15 quad-core con tecnología vSMP a 2,33 GHz y una GPU NVIDIA GK20A "Kepler" con 192 núcleos CUDA. El sistema dispone de 2 GB de memoria RAM DDR3 que comparten la CPU y la GPU, y un adaptador Realtek RTL8111/8168/8411 para la conexión con las infraestructuras 1GbE. La TDP del sistema es de 12 W.

Ambas máquinas tienen instalado el sistema operativo Linux Ubuntu 14.04 con el kernel 3.10.24, el compilador GNU gcc/g++ versión 4.8.2, OpenMPI 1.8.3, ATLAS 3.8.4, FFTW 3.2.1 y CUDA 6.0.

La política de gestión de los núcleos del procesador Tegra K1 activa permanentemente los cuatro cores principales a la máxima frecuencia.

Al igual que ocurre con los equipos CARMA1 y CARMA2, estas máquinas no presentan puertos de expansión donde conectar directamente tarjetas PCIe o PCI.

ATOM1 y **ATOM2** son máquinas multinúcleo de 64 bits equipadas con sendos procesadores Intel Atom C2750 de 8 núcleos que funcionan a 2,40 GHz y que poseen una TDP de 20 W. Cada uno dispone de 8 GB de RAM DDR3, un adaptador de Gigabit Intel I354 y un slot de expansión PCIe 3.0 x8.

El sistema está gobernado por un SO Linux CentOS 6.6 con un kernel 2.6.32-504 y el escalado de la frecuencia del procesador está controlado según el esquema `performance`. Esta distribución de Linux incluye la versión 4.4.7 del compilador GNU gcc/g++, OpenMPI 1.8.1, CUDA 6.0, ATLAS 3.8.4, FFTW 3.2.1 y OFED 3.12-1-rc4.

Plataformas con arquitecturas Intel Xeon

XEONQ1 y **XEONQ2** son dos equipos dotados cada uno con un procesador Intel Xeon X3430 de 4 núcleos a una frecuencia de 2,40 GHz y una TDP de 95 W, 8 GB de RAM DDR3, un adaptador de 1GbE Broadcom BCM5723 y un puerto PCIe 2.0 x8.

En estas máquinas se encuentra instalado un SO Linux CentOS 6.6 con kernel 2.6.32-50 configurado con el esquema `performance` para la gestión del escalado de la frecuencia del procesador. Además, la distribución del Linux comprende el compilador GNU gcc/g++ versión 4.4.7, el paquete de software OpenMPI 1.8.1, CUDA 6.0, ATLAS 3.8.4, FFTW 3.2.1 y OFED 3.12-1-rc4.

XEONS1 es una arquitectura multinúcleo equipada con un procesador Intel Xeon E5-2630L de 6 núcleos a 2,00 GHz con una TDP de 60 W, 8 GB de RAM DDR3 y un adaptador Gigabit Intel I210. La placa base dispone de 7 puertos de expansión PCIe 3.0 x16. En dos de estos puertos se encuentran ensambladas 2 GPUs NVIDIA GeForce GTX480 "Fermi" con 448 núcleos CUDA, 1.280 MB de RAM DDR3/GDDR5 y una TDP de 250 W cada una. La TDP total de la máquina es de 560 W.

El sistema operativo utilizado en XEONS1 es un Linux CentOS 6.6 con el kernel 2.6.32-50. La política de escalado de frecuencia se en-

cuentra gobernada por el esquema `performance`. El sistema incorpora el compilador GNU gcc/g++ 4.4.7, OpenMPI 1.8.1, CUDA 6.0, ATLAS 3.8.4, y OFED 3.12-1-rc4.

Plataforma	Equipos	CPU	GPU	Potencia de Diseño Térmico (TDP)
Carma	Carma1 Carma2	ARM Cortex A9 1,4 GHz 4 núcleos 2 GB de RAM DDR3	Quadro 1000M PCIe 1.0 x4 96 núcleos 2 GB de RAM DDR3	40 W
Kayla	Kayla1	ARM Cortex A9 1,4 GHz 4 núcleos 2 GB de RAM DDR3	GTX 480 PCIe 1.0 x4 488 núcleos 1.280 MB de RAM DDR3/GDDR5	10 W sin GPU 260 W con GPU
Jetson	Jetson1 Jetson2	ARM Cortex A15 2,33 GHz 4 núcleos 2 GB de RAM DDR3 compartida	GK20A PCIe 2.0 x4 192 núcleos	12 W
Atom	Atom1 Atom2	Intel Atom C2750 2,4 GHz 8 núcleos 8 GB de RAM DDR3	-	20 W
XeonQ	XeonQ1 XeonQ2	Intel Xeon X3430 2,4 GHz 4 núcleos 8 GB de RAM DDR3	-	95 W
XeonS	XeonS1	Intel Xeon E5-2630L 2,0 GHz 6 núcleos 8 GB de RAM DDR3	2 x GTX 480 PCIe 2.0 x16 488 núcleos 1.280 MB de RAM DDR3/GDDR5	60 W sin GPU 560 W con GPU

Tabla 4.1: Resumen de las principales características de las plataformas usadas en la experimentación.

De aquí en adelante, en el presente documento, se utilizan los términos Carma, Kayla, Jetson, Atom, XeonQ y XeonS para referirse a un tipo de plataforma en general. Los equipos se identifican específicamente con sus nombres correspondientes, es decir: Carma1, Carma2, Kayla1, Jetson1, Jetson2, Atom1, Atom2, XeonQ1, XeonQ2 y XeonS1.

4.1.2. Infraestructuras de red

En este apartado se describen los dispositivos que incorpora el entorno de pruebas para implementar las diferentes infraestructuras de red utilizadas en la experimentación.

Infraestructura 1GbE. Esta infraestructura está formada por los adaptadores de red 1GbE que incorporan las placas base de las plataformas que se han descrito en el apartado anterior y por un conmutador Cisco SLM2008.

Infraestructura RoCE. El entorno de pruebas consta de 2 tarjetas Mellanox MCX312A-XCBT PCIe 3.0 x8 para el acceso a infraestructuras RoCE. Estas tarjetas están soportadas únicamente por las plataformas de pruebas con procesadores Intel Atom e Intel Xeon (las plataformas `Atom`, `XeonQ` y `XeonS`). A pesar de contar con un puerto PCIe, `Kayla1` no es capaz de utilizar adaptadores RoCE ya que la plataforma no cuenta con el soporte software ni hardware adecuado.

Infraestructura iWARP. El hardware disponible para implementar la red iWARP consiste en 2 adaptadores NetEffect NE020 PCIe 1.1 x8. Estos dispositivos son incompatibles con los sistemas con arquitectura Intel Atom y ARM. Sólo es posible la conexión iWARP en los equipos `XeonQ1`, `XeonQ2` y `XeonS1`.

Infraestructura IB. Para establecer una red IB QDR (*Quad Data Rate*) se utilizan 3 adaptadores Mellanox MCX353A-QCBT PCIe 3.0 x8 y un conmutador Mellanox MTS3600. Al igual que ocurre con las infraestructuras iWARP y RoCE, la máquina `Kayla1` no soporta un adaptador que permita el acceso a la red IB.

4.1.3. Escenarios

Las plataformas hardware y las infraestructuras de red presentes en el entorno de experimentación permiten confeccionar distintas configuraciones cliente-red-servidor o CPU-GPU representativas del hardware utilizado actualmente en HPC. Para establecer estas combinaciones se han tenido en cuenta las siguientes consideraciones:

- Únicamente las plataformas `Atom`, `XeonQ` y `XeonS` presentan puertos PCIe libres donde conectar los adaptadores para las tecnologías RDMA.
- El hardware disponible para las infraestructuras iWARP y RoCE sólo permite la interconexión de dos equipos.

Configuración	Clientes	Red	Servidores	Descripción
2Carma-1GbE-1Kayla	Carma1 Carma2	1GbE	Kayla1	Dos clientes de 32 bits ARM de bajo consumo Un servidor de 32 bits ARM de bajo consumo Una GPU "Fermi"
2Carma-1GbE-2Jetson	Carma1 Carma2	1GbE	Jetson1 Jetson2	Dos clientes de 32 bits ARM de bajo consumo Dos servidores de 32 bits ARM de bajo consumo Una GPU "Kepler" integrada en cada servidor
2Carma-1GbE-1XeonS	Carma1 Carma2	1GbE	XeonS1	Dos clientes de 32 bits ARM de bajo consumo Un servidor de 64 bits Intel Xeon Una GPU "Fermi"
2Jetson-1GbE-1Kayla	Jetson1 Jetson2	1GbE	Kayla1	Dos clientes de 32 bits ARM de bajo consumo Un servidor de 32 bits ARM de bajo consumo Una GPU "Fermi"
2Jetson-1GbE-1XeonS	Jetson1 Jetson2	1GbE	XeonS1	Dos clientes de 32 bits ARM de bajo consumo Un servidor de 64 bits Intel Xeon Una GPU "Fermi"
2Atom-1GbE-2Carma	Atom1 Atom2	1GbE	Carma1 Carma2	Dos clientes de 64 bits Atom de bajo consumo Dos servidores de 32 bits ARM de bajo consumo Una GPU "Fermi" integrada en cada servidor
2Atom-1GbE-1Kayla	Atom1 Atom2	1GbE	Kayla1	Dos clientes de 64 bits Atom de bajo consumo Un servidor de 32 bits ARM de bajo consumo Una GPU "Fermi"
2Atom-1GbE-2Jetson	Atom1 Atom2	1GbE	Jetson1 Jetson2	Dos clientes de 64 bits Atom de bajo consumo Dos servidores de 32 bits ARM de bajo consumo Una GPU "Kepler" integrada en cada servidor
2Atom-1GbE-1XeonS	Atom1 Atom2	1GbE	XeonS1	Dos clientes de 64 bits Atom de bajo consumo Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"
2XeonQ-1GbE-2Carma	XeonQ1 XeonQ2	1GbE	Carma1 Carma2	Dos clientes de 64 bits Intel Xeon Dos servidores de 32 bits ARM de bajo consumo Una GPU "Fermi" integrada en cada servidor
2XeonQ-1GbE-1Kayla	XeonQ1 XeonQ2	1GbE	Kayla1	Dos clientes de 64 bits Intel Xeon Un servidor de 32 bits ARM de bajo consumo Una GPU "Fermi"
2XeonQ-1GbE-2Jetson	XeonQ1 XeonQ2	1GbE	Jetson1 Jetson2	Dos clientes de 64 bits Intel Xeon Dos servidores de 32 bits ARM de bajo consumo Una GPU "Kepler" integrada en cada servidor
2XeonQ-1GbE-1XeonS	XeonQ1 XeonQ2	1GbE	XeonS1	Dos clientes de 64 bits Intel Xeon Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"

Tabla 4.2: Configuraciones para la evaluación de aplicaciones aceleradas con GPUs remotas a través de tecnologías de red 1GbE.

- Las plataformas **Atom** del entorno de pruebas no soportan los RNICs (*RDMA Network Interface Controller*s) que permiten acceder a la infraestructura de red iWARP.
- Cuando las plataformas **Carma** y **Kayla** actúan como cliente, no existe ninguna diferencia, ya que ambos sistemas incluyen el mismo tipo de procesador. Desde el punto de vista de la potencia y el consumo, cuando la GPU de **Carma** no es utilizada, no afecta prácticamente a

Configuración	Clientes	Red	Servidores	Descripción
2Atom-IB-1XeonS	Atom1 Atom2	IB	XeonS1	Dos clientes de 64 bits Atom de bajo consumo Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"
2XeonQ-IB-1XeonS	XeonQ1 XeonQ2	IB	XeonS1	Dos clientes de 64 bits Intel Xeon Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"
1XeonQ-rWARP-1XeonS	XeonQ1	iWARP	XeonS1	Un cliente de 64 bits Intel Xeon Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"
1Atom-RoCE-1XeonS	Atom1	RoCE	XeonS1	Un cliente de 64 bits Atom de bajo consumo Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"
1XeonQ-RoCE-1XeonS	XeonQ1	RoCE	XeonS1	Un cliente de 64 bits Intel Xeon Un servidor de 64 bits Intel Xeon Dos GPU "Fermi"

Tabla 4.3: Configuraciones para la evaluación de aplicaciones aceleradas con GPUs remotas a través de tecnologías de interconexión RDMA.

estos dos factores. Por lo tanto, para simplificar los experimentos se utilizará la plataforma Carma en representación de ambas.

- Las plataformas Jetson integran una GPU local cuya potencia de cálculo es superior a la que puede proporcionar de forma remota la GPU de Carma. Incluso desde la perspectiva energética, tampoco se obtiene ningún beneficio, ya que la TDP de Carma es de 40 W frente a los 12 W de Jetson. Como resultado, las configuraciones donde las plataformas Carma actúan como servidores de clientes Jetson pueden ser descartadas.

- Tal como se indica en [104], en las configuraciones híbridas con plataformas de 32 y 64 bits, el software de rCUDA y las aplicaciones que acceden a la GPU remota deben ser compilados para arquitecturas de 32 bits, ya que rCUDA no soporta las transferencias entre clientes y servidores de distintas arquitecturas.

- La biblioteca CUDA para arquitecturas de 32 bits no permite el acceso a múltiples GPUs, por lo que los servidores rCUDA de 32 bits sólo ofrecen una GPU a los clientes de 32 bits.

En las Tablas 4.2 y 4.3 se pueden observar las configuraciones escogidas para efectuar la evaluación de la virtualización de GPUs sobre las principales tecnologías HPC, incluyendo las plataformas de bajo consumo

Configuración	Equipos	Descripción
2Carma-Local	Carma1 Carma2	Plataforma de 32 bits ARM de bajo consumo Una GPU "Fermi" integrada en cada equipo
1Kayla-Local	Kayla1	Plataforma de 32 bits ARM de bajo consumo Una GPU "Fermi"
2Jetson-Local	Jetson1 Jetson2	Plataforma de 32 bits ARM de bajo consumo Una GPU "Kepler" integrada en cada equipo
1XeonS-Local	XeonS1	Plataforma de 64 bits Intel Xeon Dos GPU "Fermi"

Tabla 4.4: Configuraciones para la evaluación de aplicaciones aceleradas con GPUs locales.

energético. En todos los casos, varias aplicaciones de prueba acceden a las GPUs remotas a través de una infraestructura de red utilizando los diferentes módulos de comunicación de rCUDA, incluyendo las soluciones completas descritas en el Capítulo 3.

Por último, en la Tabla 4.4 se enumeran los escenarios en los que las aplicaciones de prueba acceden a las GPUs locales usando la API de CUDA. Estas configuraciones están formadas exclusivamente por las plataformas que disponen de GPUs locales como Carma, Jetson, Kayla y XeonS. Su finalidad es la ejecución tradicional (sobre la GPU local) de las aplicaciones de prueba y de este modo obtener valores de referencia que permitan contrastar los resultados de las pruebas realizadas sobre los escenarios con virtualización de GPUs recogidos en las Tablas 4.2 y 4.3.

4.2. Caracterización de las transferencias a GPUs remotas

En esta sección se evalúa la solución de virtualización de rCUDA en los HPCCs actuales a través de un análisis experimental del rendimiento de las transferencias entre la memoria principal del cliente y la memoria de la GPU remota.

Para este propósito se ha ejecutado el benchmark `bandwidthTest` con el modo de funcionamiento `shmoo` sobre cada uno de los escenarios de virtualización (Tablas 4.2 y 4.3).

El programa `bandwidthTest` es un microkernel distribuido por NVIDIA en el kit de desarrollo de CUDA [81], que mide el rendimiento

de las transferencias síncronas (utilizan búferes con memoria paginable) y asíncronas (emplean búferes con memoria no paginable o *pinned memory*) desde la memoria principal de la CPU a la memoria de la GPU y viceversa. En particular, el modo de funcionamiento `shmoo` efectúa una evaluación intensiva del ancho de banda de las transferencias mediante 10 operaciones de copia de memoria consecutivas entre la CPU y GPU, seguidas de una sincronización en el caso de las transferencias asíncronas. Este test se realiza para un conjunto de datos que va desde 1 KB a 64 MB.

El benchmark se ha ejecutado en un sólo cliente/nodo que accede a una única GPU conectada a un servidor/nodo utilizando el software intermediario de la solución de virtualización de rCUDA en cada una de las configuraciones presentadas en la sección anterior.

Con el objetivo de valorar todas las alternativas que ofrece rCUDA, la ejecución se ha repetido con todos los módulos de comunicación que soportan el sistema de interconexión usado entre el cliente y el servidor (incluyendo los diseñados en el Capítulo 3).

Por otro lado, a fin de presentar los resultados obtenidos en su contexto, se ha ejecutado `bandwidthTest` sobre las configuraciones que ofrecen acceso a la GPU local (Tabla 4.4) y se ha determinado el ancho de banda disponible entre la CPU y la GPU.

En las configuraciones hardware con infraestructuras 1GbE (Tabla 4.2) se ha utilizado la versión 3.6.2 de la herramienta NetPIPE (*Network Protocol Independent Performance Evaluator*) para medir el rendimiento de las comunicaciones entre los extremos. Esta aplicación ejecuta el test de "ping-pong" sobre un rango de tamaños de mensaje para evaluar las prestaciones de una red TCP/IP [105, 106].

Por lo que se refiere a los escenarios con tecnologías RDMA (Tabla 4.3), se ha recurrido a las herramientas `ib_send_bw` y `ib_write_bw` para obtener el ancho de banda de la conexión entre el nodo cliente y el nodo servidor. Ambas aplicaciones forman parte del paquete de utilidades `perftest` [107], compuesto por una colección de programas diseñados con el propósito de realizar pruebas de rendimiento de las tecnologías RDMA. En particular, estos dos programas siguen el modelo cliente-servidor para calcular el ancho de banda de la transferencias usando el modo de transporte RC. La diferencia entre ambos es que el primero utiliza operaciones de semántica de canal mientras que el segundo ejecuta operaciones de

semántica de memoria para realizar las transferencias.

Por último, con el objetivo de reducir la variabilidad de los resultados, todos los experimentos se han efectuado 5 veces. Además, siguiendo las indicaciones descritas en [108], sólo se ha tenido en cuenta el valor máximo del rendimiento en el análisis.

4.2.1. Resultados

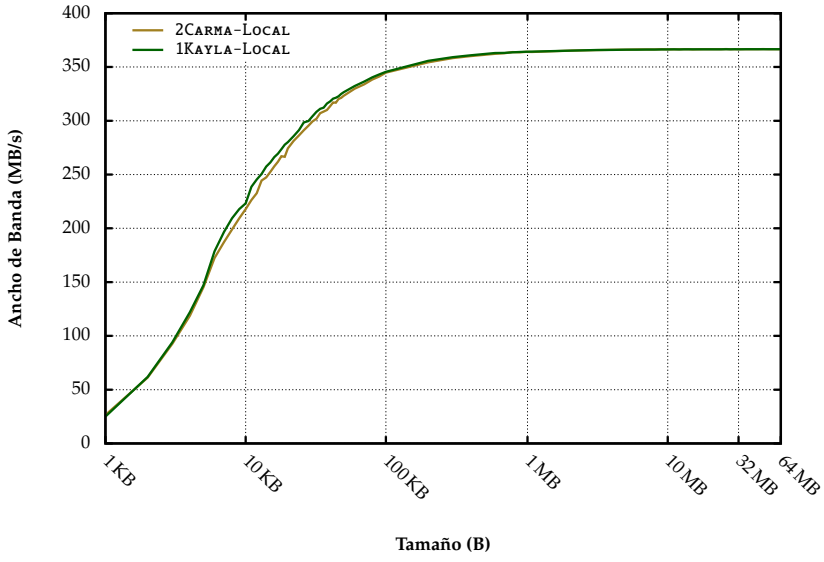
A continuación se representan los resultados recogidos al realizar las pruebas descritas. Si bien es cierto que las transferencias de CUDA obtienen diferentes rendimientos según su dirección y el tipo de memoria de los búferes utilizados, por claridad, las figuras mostradas se limitan a los valores obtenidos para las transferencias entre el cliente/*host* y el servidor/dispositivo usando memoria no paginable, siendo éstas las que mayor rendimiento proporcionan.

Escenarios con una GPU local

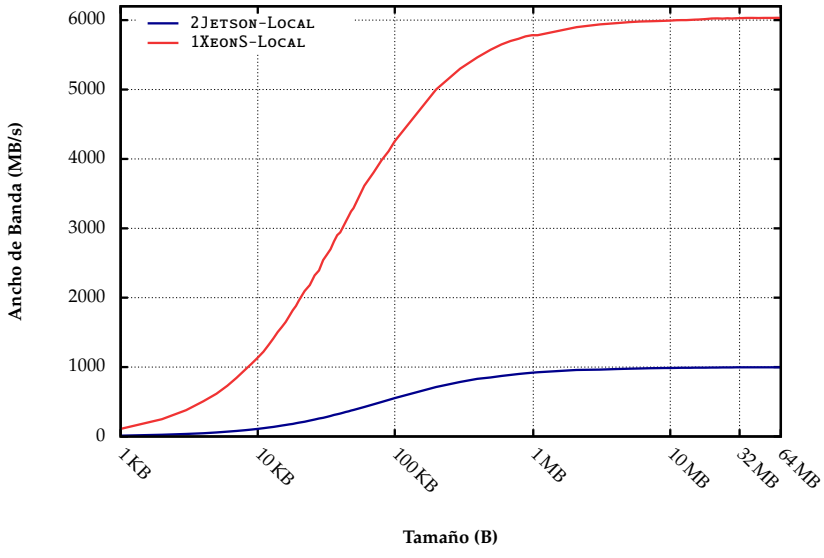
Como referencia, primero se ha realizado la evaluación de las transferencias entre la CPU y una GPU local. Para ello se ha ejecutado la aplicación `bandwidthTest` sobre las configuraciones recogidas en la Tabla 4.4. Los resultados obtenidos se presentan en la Figura 4.1.

Al comparar los resultados de las configuraciones formadas por plataformas con arquitecturas ARM, se observa como las configuraciones 2Carma-Local y 1Kayla-Local obtienen el mismo rendimiento a pesar de que 1Kayla-Local dispone de una GPU con mayor potencia. Este resultado se debe a que el ancho de banda de las transferencias se encuentra condicionado por la velocidad del bus PCIe que comunica la CPU con la GPU, que en este caso es la misma. A su vez, como se muestra en la Figura 4.1, la configuración 2Jetson-Local es capaz de alcanzar 2,73 $\times$ el rendimiento proporcionado por 2Carma-Local/1Kayla-Local. Sin embargo, en la Figura 4.2 se observa que 2Carma-Local/1Kayla-Local presenta un rendimiento mayor para tamaños menores a 50 KB, que puede ser de hasta un factor de 2,19 $\times$.

Por otro lado, es evidente que la configuración que mejores prestaciones ofrece es 1XeonS-Local, con un ancho de banda máximo de 6.032,80 MB/s para tamaños mayores a 10 MB, que resulta 16,75 $\times$ y 6,04 $\times$ más rápido



(a) Configuraciones 2Carma-Local y 1Kayla-Local



(b) Configuraciones 2Jetson-Local y 1XeonS-Local

Figura 4.1: Rendimiento de las transferencias desde la CPU a la GPU local utilizando búferes con memoria no paginable.

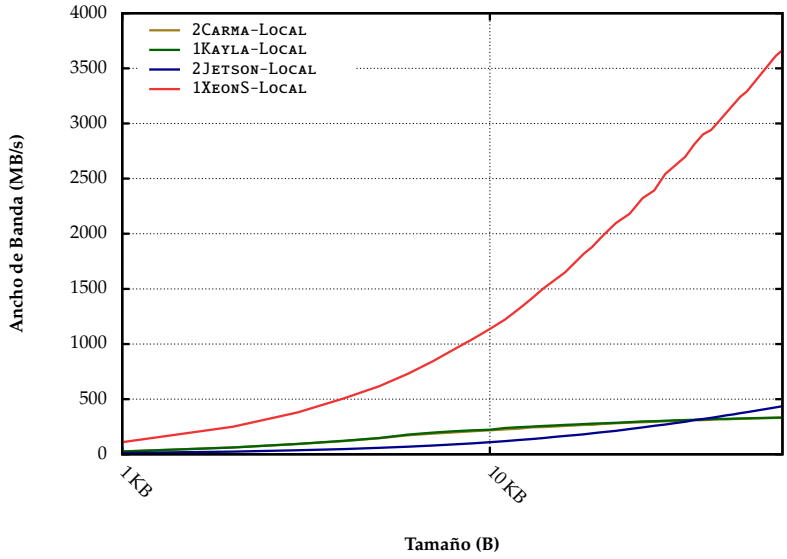


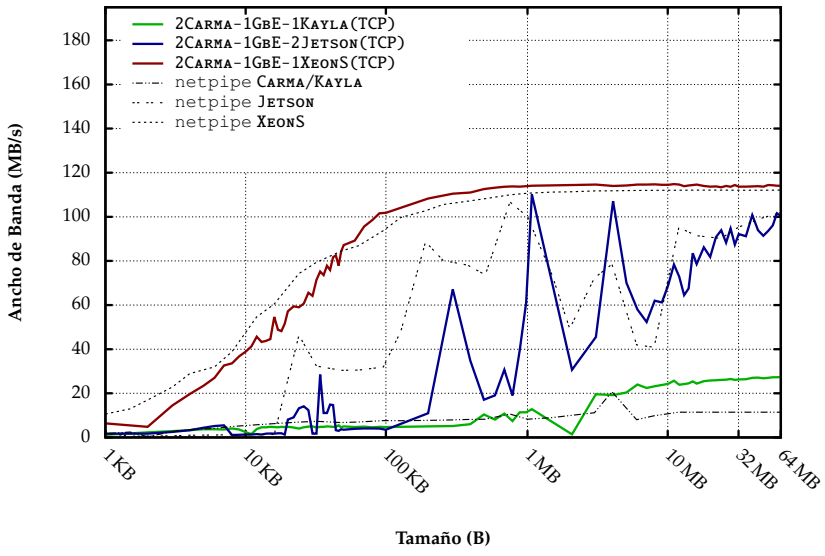
Figura 4.2: Detalle del rendimiento de las transferencias desde la CPU a la GPU local utilizando búferes con memoria no paginable.

que las configuraciones 2Carma-Local/1Kayla-Local y 2Jetson-Local, respectivamente. En este caso la limitación nuevamente viene determinada por las prestaciones del bus PCIe que conecta la CPU con la GPU. En particular, el bus de la plataforma XeonS presenta un ancho de banda teórico de 8 GB/s, mientras que para Jetson es 2 GB/s, y Carma/Kayla disponen de 1 GB/s.

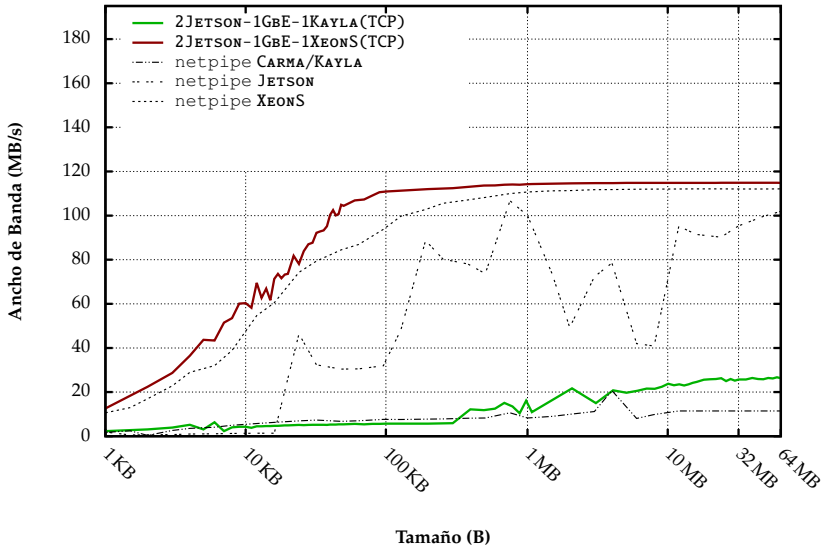
Escenarios con infraestructuras de red 1GbE

El rendimiento de las transferencias entre el cliente y la GPU remota a través de una infraestructura 1GbE, como se observa en las Figuras 4.3 y 4.4, presenta tres tendencias bien diferenciadas que son totalmente independientes de la plataforma cliente y que, sin embargo, guardan relación con la plataforma servidor utilizada:

- En primer lugar, destaca en todos los gráficos de las Figuras 4.3 y 4.4 el rendimiento de las transferencias en las configuraciones donde el servidor rCUDA se ejecuta en la plataforma XeonS. Concretamente, se alcanza un ancho de banda de 114,9 MB/s. Este valor se sitúa por encima de 112,08 MB/s, que es el ancho de banda proporcionado por la

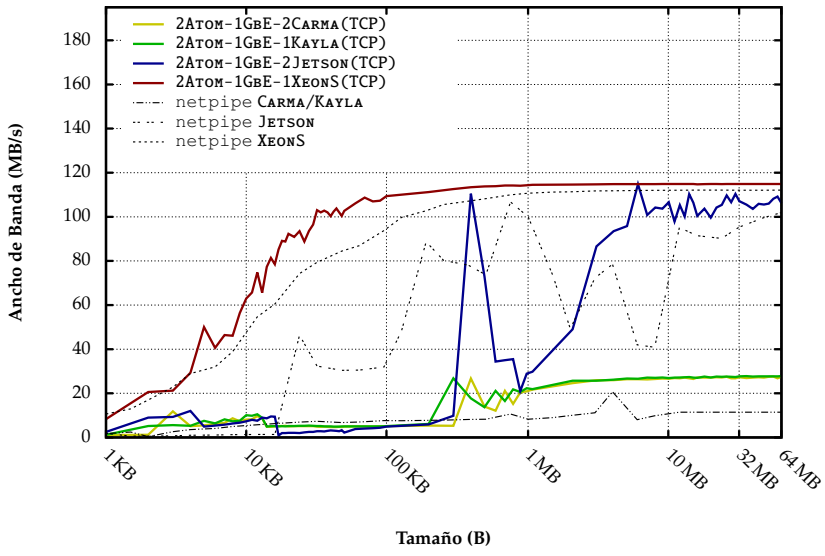


(a) Cliente Carma

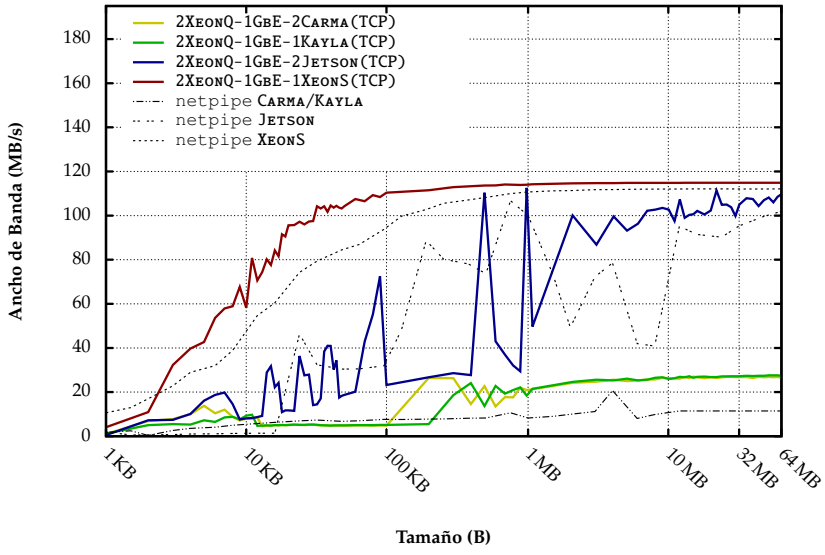


(b) Cliente Jetson

Figura 4.3: Rendimiento de las transferencias desde los clientes con arquitectura ARM hacia la GPU remota en las configuraciones con infraestructuras de red 1GbE utilizando búferes con memoria no paginable. Como referencia se muestra el ancho de banda de la conexión entre el cliente y el servidor alcanzado por la herramienta NetPIPE.



(a) Cliente Atom



(b) Cliente XeonQ

Figura 4.4: Rendimiento de las transferencias desde los clientes con arquitectura Intel Atom e Intel Xeon hacia la GPU remota en las configuraciones con infraestructuras de red 1GbE utilizando búferes con memoria no paginable. Como referencia se muestra el ancho de banda de la conexión entre el cliente y el servidor alcanzado por la herramienta NetPIPE.

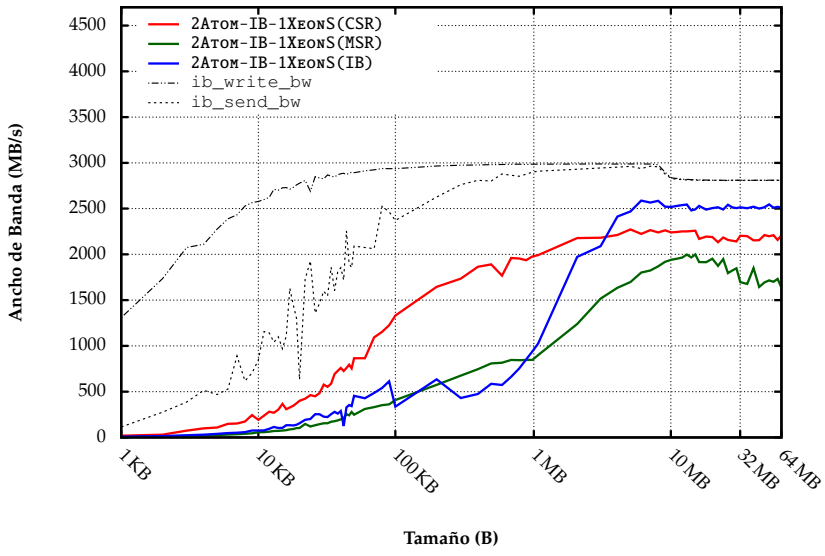
herramienta NetPIPE. Además, supone el 91,9 % del rendimiento máximo teórico de la tecnología de red 1GbE. Esto es consecuencia de la técnica de segmentación (*pipeline*) implementada por el módulo TCP.

- Por otro lado, se aprecia la existencia de una elevada dispersión en los valores de rendimiento de las transferencias de menos de 2 MB dirigidas a un servidor rCUDA en una plataforma Jetson. La desviación relativa media de los resultados en la configuración 2XeonQ-1GbE-1Jetson se encuentra en el 50,20 % para tamaños menores a 2 MB, mientras que para el resto de tamaños es del 4,21 %. Este comportamiento se presenta en todas de configuraciones donde la plataforma Jetson actúa como servidor. En cambio, en las configuraciones con red 1GbE donde XeonS, Carma o Kayla ejecutan el servidor rCUDA, la desviación relativa media es del 3,09 %. A esto se suma que la desviación relativa media de los valores medidos por la aplicación NetPIPE sobre una conexión 1GbE entre las dos plataformas Jetson es del 15,25 %, mientras que para las plataformas Carma es del 5,66 %. Así pues, se puede suponer que la oscilación en el rendimiento de las transferencias es inherente a la interfaz de red de las plataformas Jetson.

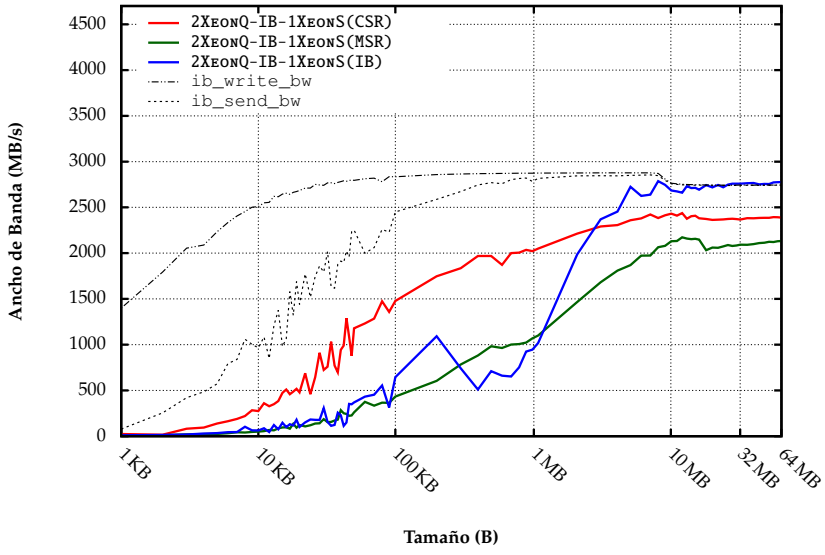
- El último aspecto relevante es el bajo rendimiento obtenido al utilizar las plataformas Carma y Kayla. Por ejemplo, 2XeonQ-1GbE-2Carma alcanza el 23,41 % del rendimiento de 2XeonQ-1GbE-1XeonS. Los resultados proporcionados por la herramienta NetPIPE para la conexión de las plataformas Carma y Kayla sobre la infraestructura de red 1GbE demuestran que la caída del rendimiento es debida a las pobres prestaciones de la interfaz de interconexión de las plataformas. En particular, las plataformas Carma y Kayla presentan un ancho de banda máximo de 13,28 MB/s, que está muy alejado del ancho de banda de 125 MB/s que se espera de una red 1GbE, lo que confirma el origen del cuello de botella.

Escenarios con tecnologías de red RDMA

Los resultados presentados en la Figura 4.5, correspondientes a las configuraciones con tecnología IB, revelan que, con independencia del cliente,



(a) Cliente Atom



(b) Cliente XeonQ

Figura 4.5: Rendimiento de las transferencias desde los clientes hacia la GPU remota en las configuraciones con infraestructuras de red IB utilizando búferes con memoria no paginable. Como referencia se muestra el ancho de banda de la conexión entre el cliente y el servidor alcanzado por las utilidades `ib_send_bw` y `ib_write_bw`.

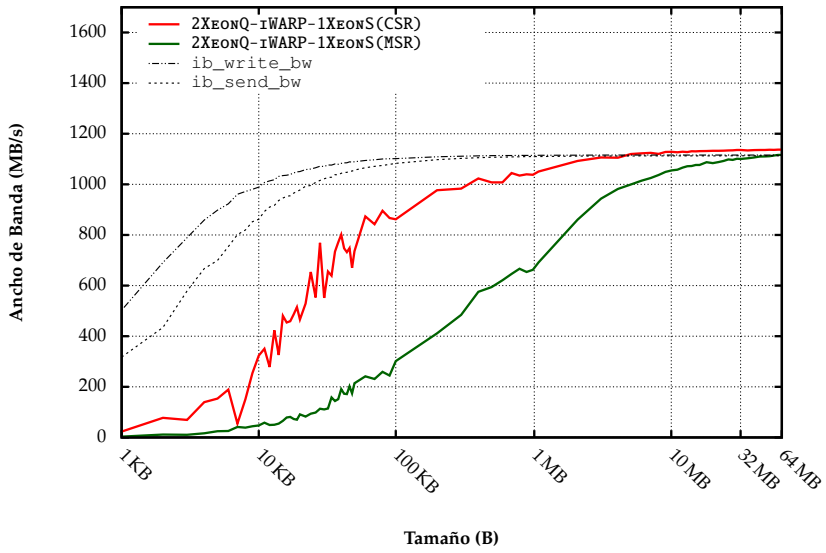
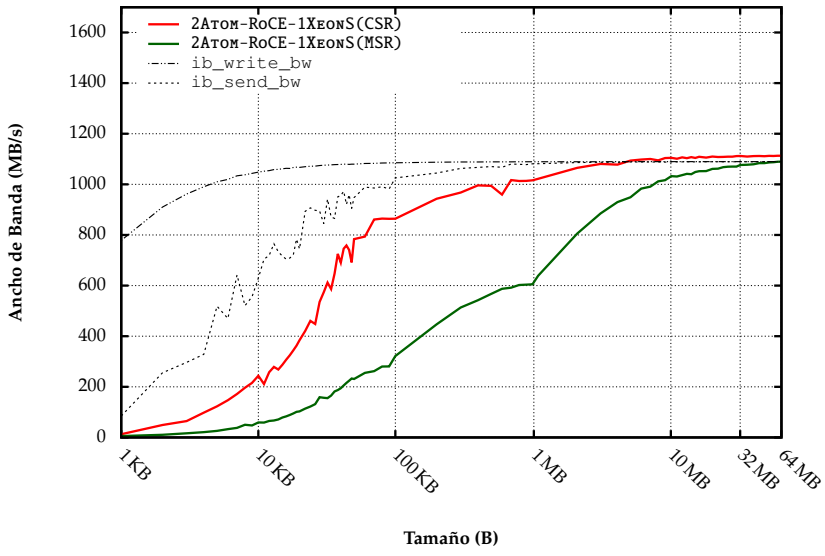


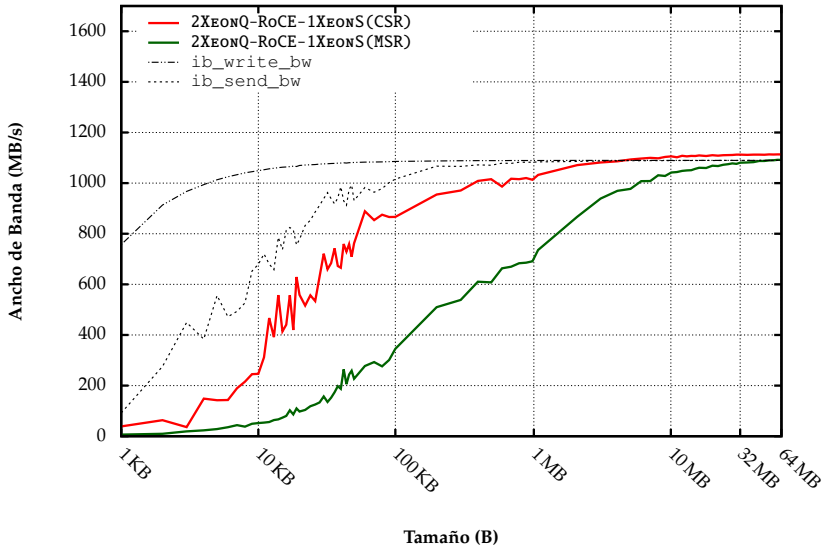
Figura 4.6: Rendimiento de las transferencias desde los clientes hacia la GPU remota en las configuraciones con infraestructuras de red iWARP utilizando búferes con memoria no paginable. Como referencia se muestra el ancho de banda de la conexión entre el cliente y el servidor alcanzado por las utilidades `ib_send_bw` y `ib_write_bw`.

el mejor rendimiento para las transferencias de tamaños menores de 1 MB se consigue empleando el módulo CSR. Concretamente, el rendimiento de esta opción es hasta un 165,93 % superior al logrado por el resto de módulos. Sin embargo a partir de 1 MB, el rendimiento alcanzado por IB crece hasta superar a CSR, siendo especialmente notable en la configuración `2XeonQ-IB-1XeonS`, donde se obtiene 2.742,42 MB/s. Este valor, que se encuentra próximo al rendimiento registrado por las herramientas `ib_send_bw` y `ib_write_bw`, supone el 68,55 % del ancho de banda máximo de la infraestructura IB QDR de 40 Gbit/s utilizada en las configuraciones.

Respecto a las configuraciones con redes iWARP y RoCE, como se aprecia en las Figuras 4.6 y 4.7, el mayor rendimiento en las transferencias entre el cliente y el servidor de rCUDA se consigue al utilizar el módulo CSR, sin importar la plataforma cliente o el tamaño de las transferencias. Más aún, CSR manifiesta un comportamiento sobresaliente para las transferencias con tamaños por encima de 2 MB, donde supera el rendimiento



(a) Cliente `Atom`



(b) Cliente `XeonQ`

Figura 4.7: Rendimiento de las transferencias desde los clientes hacia la GPU remota en las configuraciones con infraestructuras de red RoCE utilizando búferes con memoria no paginable. Como referencia se muestra el ancho de banda de la conexión entre el cliente y el servidor alcanzado por las utilidades `ib_send_bw` y `ib_write_bw`.

experimentado en las pruebas con `ib_send_bw` y `ib_write_bw`. En concreto, se alcanza 1.137,40 MB/s lo que representa el 90,99 % del ancho de banda teórico de la red.

Por último, aunque el rendimiento de rCUDA al utilizar MSR se mantiene siempre por debajo de CSR, los resultados obtenidos en las configuraciones con tecnologías iWARP y RoCE, al emplear MSR, se caracterizan por presentar una mejor desviación relativa. En particular, la desviación relativa media para transferencias con tamaños menores a 100 KB con MSR es del 3,57 % mientras que CSR posee una desviación relativa media del 14,83 %.

4.2.2. Conclusiones

Uno de los factores que pueden limitar el rendimiento de las aplicaciones híbridas que emplean aceleradores gráficos es el sobrecoste producido por las copias de los datos entre la CPU y la GPU a través del bus PCIe. Para una aplicación CUDA que debe procesar un enorme cantidad de datos ubicados en la memoria principal de la CPU, puede ocurrir que se consuma más tiempo en la transferencia de esos datos a la GPU que en el cálculo que debe realizarse.

Además, esta penalización se incrementa al utilizar la virtualización de GPU ya que el sistema de interconexión entre el cliente y el servidor inserta un nuevo nivel de retardo en las transferencias. Como resultado, se tiene que el rendimiento de las transferencias en las configuraciones con virtualización depende de los siguientes factores:

- La tasa de transferencia datos del bus PCIe que conecta la CPU con la GPU en el servidor.
- El sobrecoste introducido por el propio software de rCUDA.
- El ancho de banda del sistema de interconexión existente entre el cliente y el servidor rCUDA.

Una comparación del rendimiento de las distintas configuraciones con virtualización de GPUs frente a las transferencias sobre las mismas GPUs locales indica que, en el mejor de los casos, el ancho de banda de las transferencias hacia una GPU remota es 0,47 $\times$ veces menor. Así, por ejemplo,

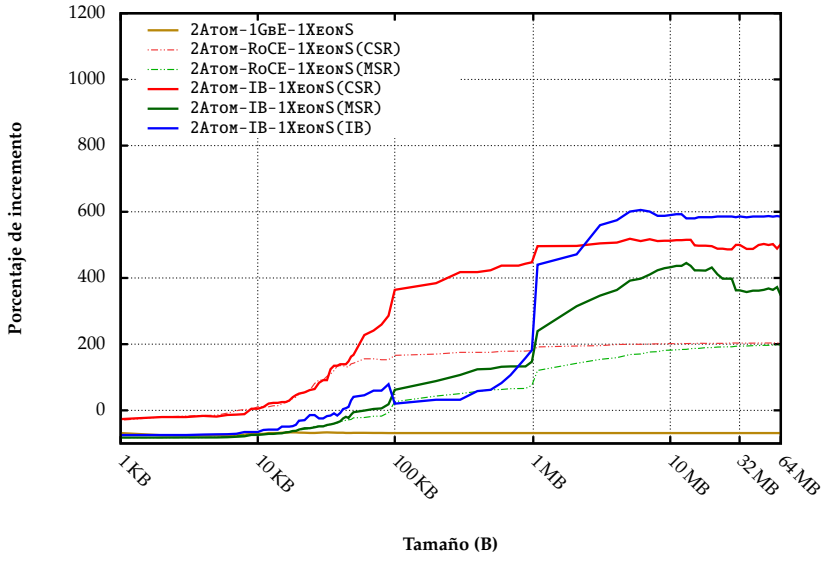
como se observa la Figura 4.8, 2XeonQ-1GbE-2Carma obtiene el 32,98 % del rendimiento de 2Carma-Local; y, como puede verse en Figura 4.10, la configuración 2XeonQ-IB-1XeonS sólo alcanza el 46,04 % del rendimiento de 1XeonS-Local. Esto sugiere que el ancho de banda del bus PCIe que conecta la GPU con la CPU del servidor no supone una limitación para las transferencias en las configuraciones con rCUDA, puesto que son valores muy alejados del rendimiento que proporciona el bus PCIe.

Por otro lado, al poner en contexto los resultados obtenidos en las configuraciones con red 1GbE, puede observarse que el rendimiento de las configuraciones supera el ancho de banda alcanzado por `NetPIPE`. Por tanto, el sobrecoste del software rCUDA no constituye una limitación. Sin embargo, este rendimiento supone el 91,9 % del ancho de banda máximo de la red de interconexión. De este modo, el rendimiento obtenido en estas configuraciones está condicionado por el rendimiento máximo de la red. A esto se añade que los experimentos revelan un importante cuello de botella en el acceso a la red por parte de los sistemas Carma y Kayla. Además, la plataforma Jetson presenta cierta inestabilidad en la tasa de transferencia de su interfaz de red, que se propaga a las transferencias entre el cliente y el servidor rCUDA.

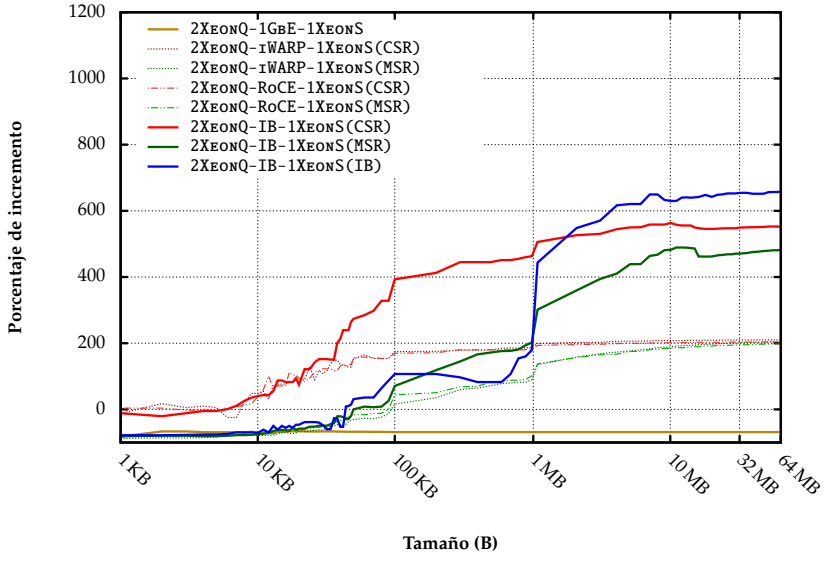
Con respecto a las configuraciones con tecnologías 10GbE (las redes iWARP y RoCE), las transferencias de menos de 2 MB están limitadas por el sobrecoste del software ya que, como se muestra en las Figuras 4.6 y 4.7, su rendimiento está alejado del registrado por `ib_send_bw` y `ib_write_bw`. Ahora bien, para tamaños de más de 2 MB, donde se alcanza el 90,99 % del ancho de banda teórico de la red 10GbE, la restricción proviene de la tasa de transferencia de la red.

En el caso de las configuraciones con infraestructura de red IB, el rendimiento de las transferencias alcanza el 70 % del ancho de banda de la red. Por tanto, el factor que limita el rendimiento de las transferencias en las configuraciones con tecnología IB es el sobrecoste introducido por el software de rCUDA.

No obstante, a pesar de esto, las transferencias en las configuraciones con virtualización de GPUs pueden ofrecen mayor rendimiento que las transferencias a GPUs locales en plataformas ARM. Por ejemplo, en la Figura 4.8, se observa que las configuraciones con redes RDMA que utilizan el módulo de comunicaciones CSR desarrollado en el presente trabajo, a partir

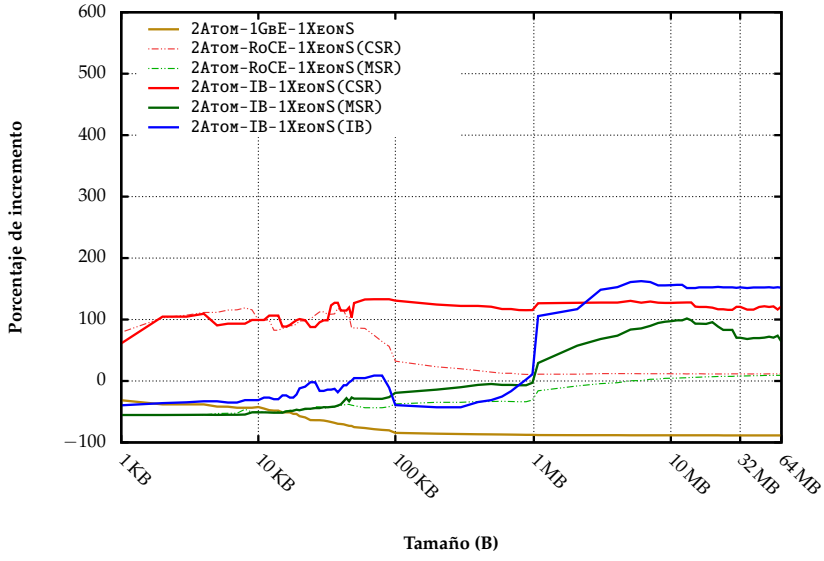


(a) Cliente ATOM

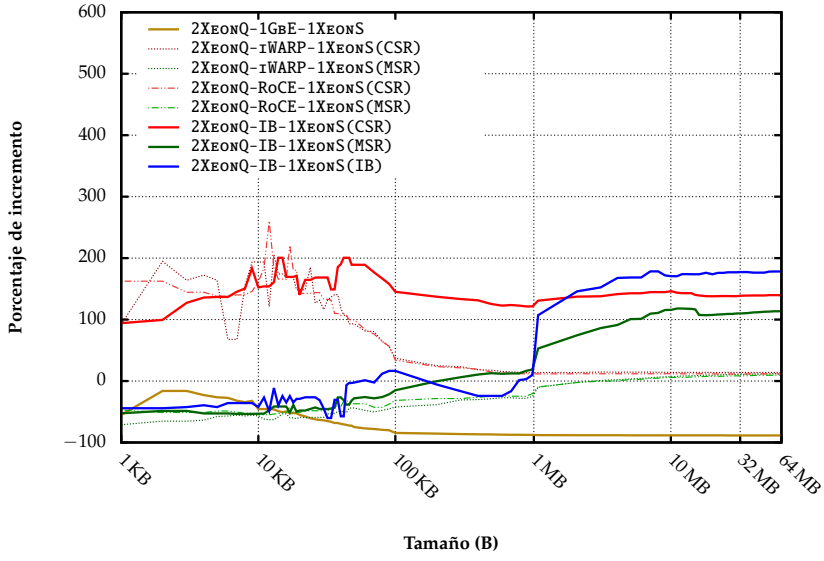


(b) Cliente XEONQ

Figura 4.8: Porcentaje de incremento del ancho de banda de las transferencias a las GPUs remotas respecto al ancho de banda obtenido por las transferencias a una GPU local en 2CARMA-LOCAL.

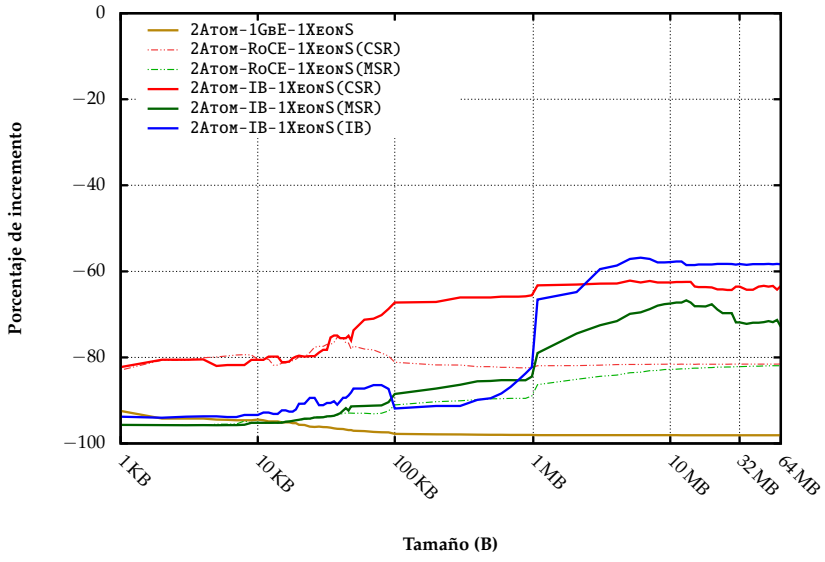


(a) Cliente ATOM

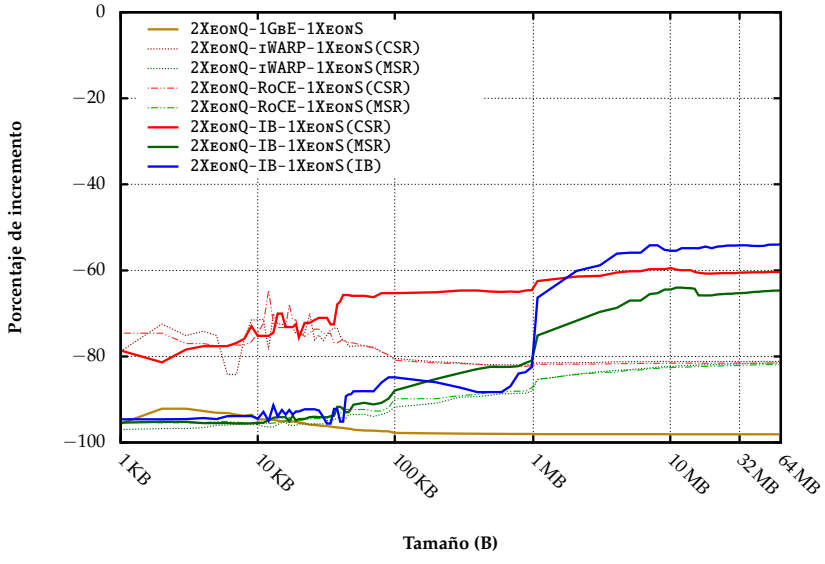


(b) Cliente XEONQ

Figura 4.9: Porcentaje de incremento del ancho de banda de las transferencias a las GPUs remotas respecto al ancho de banda obtenido por las transferencias a una GPU local en 2JETSON-LOCAL.



(a) Cliente ATOM



(b) Cliente XEONQ

Figura 4.10: Porcentaje de incremento del ancho de banda de las transferencias a las GPUs remotas respecto al ancho de banda obtenido por las transferencias a una GPU local en 1XEONS-LOCAL.

de tamaños de 10 KB, igualan y superan en rendimiento a 2Carma-Local o 1Kayla-Local. En el caso particular de la configuración 2XeonQ-IB-1XeonS, en las transferencias de más de 2 MB se aumenta el ancho de banda de 2Carma-Local y 1Kayla-Local un 551,3 %.

Al mismo tiempo, en la Figura 4.9 se aprecia que las configuraciones con infraestructuras RDMA, mediante el módulo CSR, consiguen más del 100 % del rendimiento de la configuración 2Jetson-Local. Concretamente, con la tecnología de interconexión IB, el rendimiento medio se incrementa un 100 %, alcanzando puntualmente en la configuración 2XeonQ-IB-1XeonS un 200,50 % al ejecutar transferencias de 40 KB.

En definitiva, las configuraciones formadas por plataformas con procesadores de propósito general Intel Xeon y procesadores de bajo consumo Intel Atom presentan un rendimiento muy similar cuando acceden a una GPU remota a través de las infraestructuras 1GbE o 10GbE. Sin embargo, al realizar transferencias a la GPU remota a través de redes IB la diferencia máxima entre el ancho de banda de ambas plataformas puede alcanzar el 7,9 %. Por otro lado, el módulo CSR, desarrollado en este trabajo, permite que las plataformas XeonQ y Atom realicen transferencias entre el cliente y la GPU remota a través de tecnologías RDMA con un rendimiento cercano, o incluso superior, al obtenido entre el *host* y la GPU local en las configuraciones basadas en plataformas Jetson, Kayla y Carma. En la siguiente sección de este capítulo se examinará si estas plataformas, en combinación con rCUDA, suponen una alternativa factible a las plataformas híbridas de bajo consumo con arquitectura ARM.

4.3. Aspectos generales de la evaluación de las aplicaciones aceleradas con GPUs

4.3.1. Metodología de experimentación

La solución de virtualización de rCUDA, junto con los módulos desarrollados en este trabajo, permiten aprovechar las posibilidades de ahorro energético que pueden conseguirse al aplicar la virtualización de GPUs sobre las actuales tecnologías de los HPCC cuando se ejecutan las aplicaciones que requieren GPGPU. Con el propósito de explorar estas capacidades se ha llevado a cabo un análisis experimental del rendimiento y del consumo

de la ejecución de un benchmark y tres aplicaciones paralelas de producción sobre las configuraciones (CPU-GPU) establecidas en el Apartado 4.1.3, donde clientes y servidores rCUDA se han distribuido en nodos con distintos tipos de procesadores y de GPUs, respectivamente.

Cabe señalar que los benchmarks son herramientas diseñadas para imitar un tipo particular de carga de trabajo. Su finalidad es saturar partes del sistema y obtener una medida que permita establecer una pauta en el rendimiento de las aplicaciones con características similares. En la experimentación, se ha utilizado el benchmark HPL-Fermi (*High-Performance Linpack benchmark for Fermi*) con el objetivo de contrastar el rendimiento computacional y el consumo energético de los sistemas con virtualización de GPUs (Tablas 4.2 y 4.3) con las ejecuciones aceleradas con las GPUs locales (Tabla 4.4).

Por otro lado, se emplea normalmente el término aplicación de producción para referirse a aquellas aplicaciones paralelas que tienen un uso directo en la investigación o en la industria, como por ejemplo las aplicaciones de dinámica de fluidos, dinámica molecular, biotecnología, prospección geológica, etc. Básicamente consisten en programas que implementan modelos matemáticos o simulan fenómenos físicos y que requieren gran potencia de cálculo para obtener resultados en un espacio de tiempo razonable. Al utilizar aplicaciones de producción en la experimentación se consigue reflejar cargas de trabajo reales, por lo que es posible obtener una información más realista del rendimiento y de la eficiencia de las configuraciones hardware. Para el presente estudio se han escogido las aplicaciones de producción LAMMPS (*Large-scale Atomic/Molecular Massively Parallel Simulator*), CUDASW++ (*CUDA GPU accelerated Smith-Waterman algorithm*) y GPU-BLAST (*GPU Basic Local Alignment Search Tool*). Estas aplicaciones tienen especial relevancia debido a su papel en la investigación científica actual [80].

En todos los casos, se ha establecido la evaluación del rendimiento junto con la medición del consumo energético como principales factores de la experimentación. Dicho rendimiento se ha calculado a través de las métricas facilitadas directamente por las aplicaciones o mediante las medidas de tiempo efectuadas sobre el propio código al principio y al final de la ejecución. En cambio, como se describe en [101], para la determinación de

la energía y la potencia eléctrica consumida existen varias posibilidades:

- Ejecutar herramientas software sobre la propia máquina para obtener la energía consumida por diversos componentes del equipo utilizando los contadores internos del sistema descritos en [109, 110]. Un ejemplo de estas utilidades se puede encontrar en [111-114].

- Medir individualmente el uso de energía de los componentes del equipo, como son los procesadores, la memoria y los periféricos, interceptando, mediante el uso de un amperímetro, los cables procedentes de la fuente de alimentación o PSU (*Power Supply Unit*) del equipo al componente [115, 116].

- Emplear dispositivos externos que facilitan la medida de la potencia eléctrica entre la toma de corriente y la PSU del equipo [12, 70].

Debido a la falta de herramientas software de medición de consumo disponibles para ciertas arquitecturas, y a que algunas de las plataformas integran las líneas de alimentación de los componentes sobre la placa base, se ha escogido la última alternativa para capturar los consumos energéticos de los nodos durante la experimentación.

Para ello se han utilizado dos medidores de potencia Watts Up? PRO [100]. Este tipo de dispositivo es capaz de proporcionar medidas de 19 atributos eléctricos (vatios instantáneos, vatios máximos, vatios/hora, costes eléctricos, ...) con un tiempo de muestreo de 1 segundo. El primero de los dos medidores de potencia se encarga de monitorizar los equipos que ejecutan los clientes de rCUDA. El otro dispositivo de medida lleva a cabo la misma tarea con las máquinas donde se ubican los servidores rCUDA.

Cada uno de los medidores transmite las muestras obtenidas a través de una interfaz USB a un equipo externo que ejecuta un programa de adquisición de datos utilizando la API `iec` [117] para interpretar los datos recibidos y registrarlos en un fichero cada segundo.

Un script ubicado en ese mismo equipo externo se encarga de arrancar los servidores rCUDA, iniciar la medición de consumo y enviar la orden de ejecución de la carga de trabajo a los equipos correspondientes. Una vez se completa la ejecución, el script finaliza el programa de captura y recoge el registro de datos eléctricos y las métricas ofrecidas por la aplicación.

Con el propósito de limitar el ruido, cada experimento se ha repetido 6 veces, y para el análisis de los resultados, únicamente se ha tenido en consideración la mejor de las 5 últimas ejecuciones en cuanto a rendimiento [108]. La primera ha sido descartada directamente porque recoge el periodo de calentamiento de la configuración hardware.

Por último, a fin de evitar que las condiciones ambientales del entorno de experimentación afecten al consumo energético de los equipos informáticos [7, 9, 118] se han controlado los siguientes parámetros:

Temperatura. En [8] se indica que la temperatura tiene una gran influencia sobre la potencia consumida. En general, cuanto menor es la temperatura del aire donde se encuentra el equipo, menor es la potencia necesaria para operar el equipo. Por esta razón se ha intentado que se mantenga constante durante todo el periodo de experimentación.

Humedad. Los resultados expuestos en [11] indican que la humedad tiene un impacto insignificante sobre los consumos de los equipos eléctricos. En consecuencia, no se han fijado límites mínimos o máximos para la humedad relativa, salvo los necesarios para el correcto funcionamiento de los equipos.

Flujos de aire. No se han utilizado fuentes externas de ventilación que puedan enfriar los equipos durante el curso de la medición.

Atributos del suministro eléctrico. El voltaje y otras características de la línea de suministro alteran la eficiencia del equipo que está conectado a ella [12]. Por esa razón se han mantenido los valores de frecuencia a 50 Hz ± 1 % y los del voltaje a 230 V ± 2 %.

4.3.2. Métricas

A continuación se definen las principales medidas de rendimiento utilizadas para evaluar las prestaciones y el impacto de la virtualización de GPUs propuesta por rCUDA sobre el benchmark y las aplicaciones que se emplean como cargas de trabajo en los experimentos.

Tiempo de Ejecución o TtS (*Time to Solution*). Es el tiempo necesario para ejecutar y finalizar la carga de trabajo.

Productividad o Tpt (*Troughput*). Permite determinar el rendimiento y se calcula como el trabajo útil realizado por unidad de tiempo. Dependiendo de la unidad establecida para medir el trabajo útil se tiene que:

- Cuando se dispone de información relativa al número de operaciones en coma flotante efectuadas para alcanzar la solución del problema (N_{flops}), la productividad se calcula como:

$$Tpt = \frac{N_{flops}}{TtS} \tag{4.1}$$

- Si el trabajo útil se concreta como el número de instancias del problema finalizadas (N_{tasks}) durante el intervalo de tiempo de ejecución de la carga de trabajo (T), la productividad se mide como:

$$Tpt = \frac{N_{tasks}}{T} \tag{4.2}$$

Factor de Mejora del Rendimiento o *speed-up*. Indica la mejora del rendimiento obtenida respecto a una ejecución base. El *speed-up* es la relación entre la productividad al utilizar rCUDA (Tpt_{rCUDA}) y la productividad sobre la GPU local (Tpt_{CUDA}).

$$S_{Tpt} = \frac{Tpt_{rCUDA}}{Tpt_{CUDA}} \tag{4.3}$$

Eficiencia Paralela o PEff (*Parallel Efficiency*). La eficiencia paralela es la relación entre el *speed-up* y el número de procesos o tareas paralelas ejecutadas. Idealmente, el tiempo de ejecución de un programa se reduce de manera proporcional al número de procesos utilizados por lo que el valor de la eficiencia deseable es 1 (es el máximo teórico). En la práctica esto no sucede, ya que se produce una pérdida de eficiencia debida a la creación de procesos, las dependencias y las barreras de sincronismo, los cuellos de botella en la red de interconexión, etc.

Potencia Eléctrica Media o HLP (*Hardware Load at the Plug*). Se define en [119] como el consumo de potencia de corriente alterna (AC) de cada una de las partes de un HPCC. Esta medida ofrece información de la cantidad de potencia que debe ser suministrada por la unidad de distribución de energía, o PDU (*Power Distribution Unit*),

y el SAI (Sistema de Alimentación Ininterrumpida) a la fuente de alimentación de cada equipo informático.

Rendimiento por Vatio o PpW (*Performance per Watt*). Es la medida de la eficiencia energética del hardware de cálculo utilizado en un HPCC para una carga de trabajo específica. Cuanto más elevado es este valor más eficientemente trabaja un sistema. La métrica tiene en cuenta el hardware del HPCC, el SO y las aplicaciones que se están ejecutando [120]. Si el rendimiento máximo que alcanza una aplicación se denota como P_{max}, y la potencia media requerida por el sistema durante la ejecución con rendimiento P_{max} se define como $\overline{Pw}(P_{max})$, entonces:

$$PpW = \frac{P_{max}}{\overline{Pw}(P_{max})} \tag{4.4}$$

Habitualmente, se usa $GFLOPS$ como unidad de P_{max}, los vatios como unidad de $\overline{Pw}(P_{max})$, y en consecuencia GFLOPS/W como unidad de PpW [15]. No obstante, también es posible expresar el PpW como flops/julio, ya que 1 vatio = 1 julio/segundo.

Energía para la Solución o EtS (*Energy to Solution*). En [121] se define EtS como la energía consumida por el sistema HPC para resolver un problema utilizando una aplicación específica; es decir, es el consumo de la aplicación. La métrica está determinada por el hardware del HPCC y la aplicación [122]:

$$EtS = TtS \cdot HLP_{avg} \tag{4.5}$$

Básicamente, EtS es la particularización de la métrica FTTSE propuesta por [123]:

$$FTTSE = f(TtS) \cdot Energy \tag{4.6}$$

donde si $f(TtS) = 1$ entonces $FTTSE = EtS$.

4.4. Evaluación del consumo energético de las GPUs remotas en estado inactivo

Es importante remarcar que, desde el punto de vista del tiempo de ejecución, en general una aplicación que utiliza una GPU remota no puede

mejorar los resultados obtenidos al ejecutarla en una GPU local (con la misma configuración de GPU, bus PCIe y CUDA). Normalmente, se puede esperar que el acceso remoto introduzca un ligero sobrecoste y que, en el mejor caso, el tiempo de ejecución entre ambas configuraciones (GPU local y remota) sea muy parecido. Desde la perspectiva del consumo energético, el uso de una GPU local es más ventajoso, ya que únicamente se utiliza un nodo, mientras que en la virtualización se emplean dos equipos (cliente rCUDA y servidor rCUDA). Sin embargo, el uso de rCUDA permite construir un clúster con un pequeño número de GPUs, lo que conlleva la reducción del coste tanto energético como de adquisición con respecto a la configuración de un clúster donde todos los nodos tienen al menos una GPU, que permanece ociosa un gran porcentaje del tiempo.

Plataforma	Potencia (watt)	Plataforma	Potencia (watt)
Kayla	74	Atom	32
Carma	11	XeonQ	48
Jetson	7	XeonS	166
Kayla (sin GPU)	42	XeonS (sin GPUs)	86

Tabla 4.5: Potencia media (HLP) requerida por las plataformas de experimentación en estado ocioso.

Los grandes beneficios que un sistema con un moderado número de GPUs ofrece pueden resultar más claros si se comparan, por ejemplo, las diferencias de potencia entre el servidor **XeonS** y los clientes **XeonQ** y **Atom**. En particular, los datos de la Tabla 4.5 indican que la potencia disipada por una GPU "Fermi" en estado ocioso está comprendida entre 40 W (comparando la potencia de **XeonS** en estado ocioso sin y con GPUs) y 32 W (comparando **Kayla** en estado ocioso sin y con GPU), lo que supone un incremento significativo cuando se compara con un nodo **Atom**. De esta forma se puede considerar que la virtualización permite reducir el consumo de energía cuando el sistema está ocioso en un 50 % si se emplean plataformas basadas en Intel Atom.

4.5. Evaluación de HPL-Fermi

HPL-Fermi (*High-Performance Linpack benchmark for Fermi*) es una implementación del HPL (*High-Performance Linpack benchmark*) realizada por NVIDIA [124]. A su vez, HPL es un benchmark LINPACK [125] para

sistemas paralelos de memoria distribuida que genera y calcula la solución de sistemas lineales densos de ecuaciones del tipo $A \cdot x = b$ de orden n [126]. HPL-Fermi pretende maximizar el rendimiento de HPL mediante la utilización de GPUs con arquitectura "Fermi" [127].

El código fuente de HPL-Fermi está escrito en C y emplea doble precisión en las operaciones de coma flotante. Se caracteriza por solapar el cálculo con la transferencia de datos utilizando las herramientas que proporciona la API de CUDA, como por ejemplo los *streams* de CUDA y la memoria no paginable. Además, efectúa llamadas a la API de BLAS (*Basic Linear Algebra Subprograms*) para las operaciones básicas de álgebra lineal y a una API de MPI (*Message Passing Interface*) para las comunicaciones entre las tareas o procesos. En concreto, en el presente estudio se ha utilizado el ejecutable de la versión 15 de HPL-Fermi enlazado con las bibliotecas de software OpenMPI, CUDA y BLAS de ATLAS [128] disponibles en cada una de las plataformas de la configuración.

Asimismo, HPL-Fermi ha sido diseñado para ser un programa altamente escalable. En este sentido, la distribución de la matriz de coeficientes entre los procesos de cálculo garantiza la escalabilidad del programa y el equilibrio de la carga. La matriz, de tamaño $n \times (n+1)$, se distribuye en bloques de tamaño $b \times b$ de forma cíclica, en una malla bidimensional de $p \times q$ procesos, donde cada proceso utiliza un sólo núcleo del procesador de la CPU y una GPU de forma exclusiva [124]. El código de HPL-Fermi permite, a través de un fichero de configuración, ajustar los parámetros que determinan el tamaño de la matriz de coeficientes (n) y el esquema de distribución de los datos (b, q y p).

En la experimentación, el benchmark se ha configurado para resolver sistemas lineales de orden 1.024 , 2.048 , 4.096 , 8.192 y 16.384 ; con tamaños de bloque 512 , 768 , 1.024 y 1.280 para cada tamaño de matriz. El resto de parámetros de configuración se ha mantenido constante.

Con el propósito de evitar que las ejecuciones con tamaños menores finalicen sin obtener suficientes muestras de consumo, se ha modificado el código de HPL-Fermi para repetir el proceso de resolución durante al menos 60 segundos.

Los resultados de rendimiento presentados se corresponden con los proporcionados directamente por el ejecutable al finalizar cada una de las pruebas utilizando el tamaño de bloque óptimo. En los casos donde el

proceso de resolución del sistema de ecuaciones se repite varias veces, el programa muestra el valor de la mejor iteración.

Por otra parte, las pruebas preliminares realizadas han demostrado que las plataformas con arquitecturas de 32 bits no soportan el benchmark HPL-Fermi. Por ello se ha descartado la evaluación de las configuraciones con plataformas de 32 bits, es decir, todas aquellas que disponen de plataformas con arquitectura ARM.

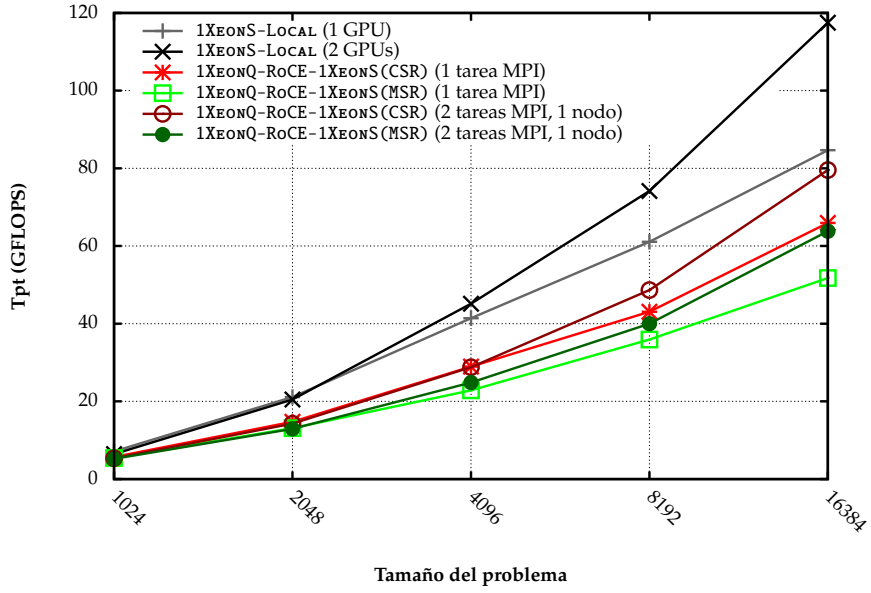
4.5.1. Aceleración con una GPU

En primer lugar se han evaluado el rendimiento computacional y la eficiencia energética de las configuraciones en las que plataformas XeonQ y Atom ejecutan el benchmark utilizando una malla ($q \times p$) de 1×1 tareas MPI, donde una tarea accede a una GPU remota. Como referencia, los resultados se han comparado con la ejecución acelerada con una GPU local sobre la configuración 1XeonS-Local.

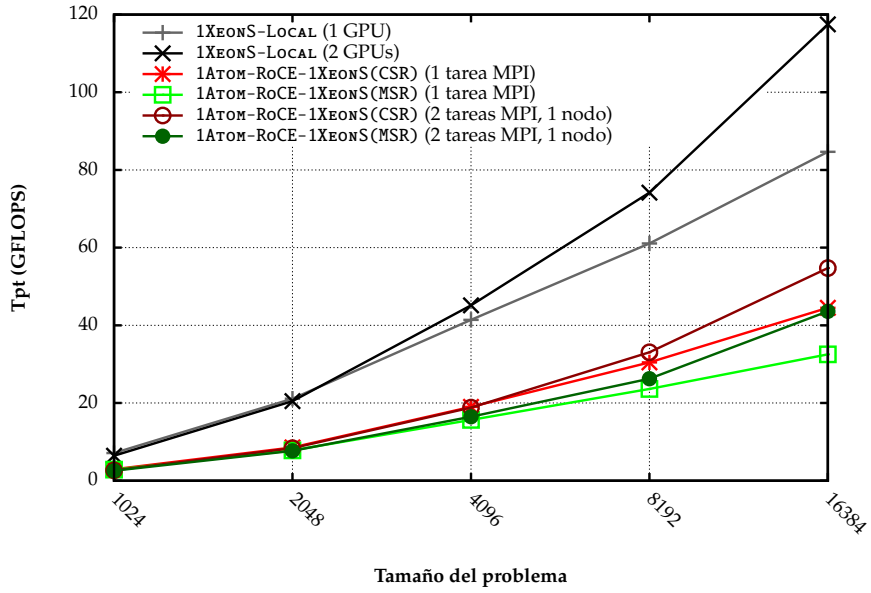
Escenario	Tamaño del problema									
	1.024		2.048		4.096		8.192		16.384	
	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)
1XeonS-Local	0,10	8	0,27	23	1,11	107	6,00	639	34,6	3.984
2XeonQ-IB-1XeonS(CSR)	0,12	10	0,33	34	1,33	151	7,37	889	40,1	5.129
2XeonQ-IB-1XeonS(MSR)	0,12	11	0,36	33	1,53	159	8,68	954	45,4	5.209
2XeonQ-IB-1XeonS(IB)	0,15	13	0,44	40	1,60	157	8,11	864	40,0	4.639
2Atom-IB-1XeonS(CSR)	0,23	17	0,62	46	2,21	179	11,21	999	63,1	5.622
2Atom-IB-1XeonS(MSR)	0,24	18	0,66	49	2,47	197	13,87	1.132	68,9	5.856
2Atom-IB-1XeonS(IB)	0,27	20	0,73	56	2,52	202	11,87	1.010	62,4	5.329
1XeonQ-RoCE-1XeonS(CSR)	0,13	12	0,39	38	1,58	169	8,51	985	44,5	5.434
1XeonQ-RoCE-1XeonS(MSR)	0,13	12	0,44	40	2,01	194	10,21	1.058	56,7	6.132
1Atom-RoCE-1XeonS(CSR)	0,24	17	0,67	47	2,41	186	12,04	993	65,9	5.572
1Atom-RoCE-1XeonS(MSR)	0,25	18	0,73	49	2,93	227	15,52	1.171	80,6	7.003
1XeonQ-iWARP-1XeonS(CSR)	0,13	13	0,39	39	1,57	171	8,45	1.014	44,2	5.529
1XeonQ-iWARP-1XeonS(MSR)	0,14	13	0,44	43	2,01	205	10,20	1.087	56,5	6.241
2XeonQ-1GbE-1XeonS(TCP)	0,27	28	1,34	144	6,60	717	33,78	3.495	216,1	22.108
2Atom-1GbE-1XeonS(TCP)	0,41	35	1,72	147	8,08	704	40,38	3.349	248,9	20.598

Tabla 4.6: Tiempo (TtS) y energía (EtS) requeridos para resolver el sistema de ecuaciones de HPL-Fermi utilizando 1 tarea MPI.

Los resultados recogidos en la Tabla 4.6 indican que las configuraciones con plataformas Atom presentan un tiempo de ejecución mayor que las configuraciones que utilizan plataformas XeonQ. Esto es debido a que los



(a) Cliente XeonQ



(b) Cliente Atom

Figura 4.11: Comparativa del rendimiento (Tpt) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red RoCE frente a las ejecuciones aceleradas con las GPUs locales.

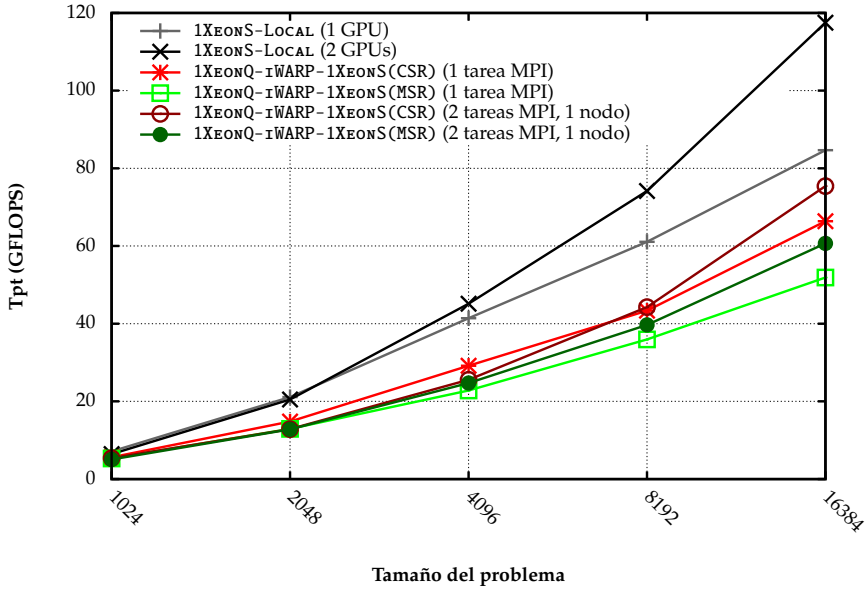


Figura 4.12: Comparativa del rendimiento (Tpt) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red iWARP frente a las ejecuciones aceleradas con las GPUs locales.

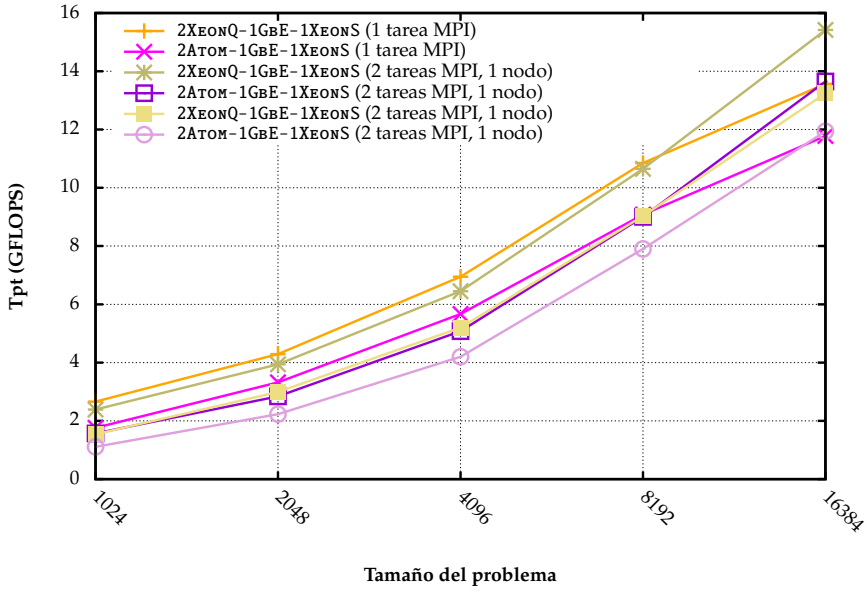
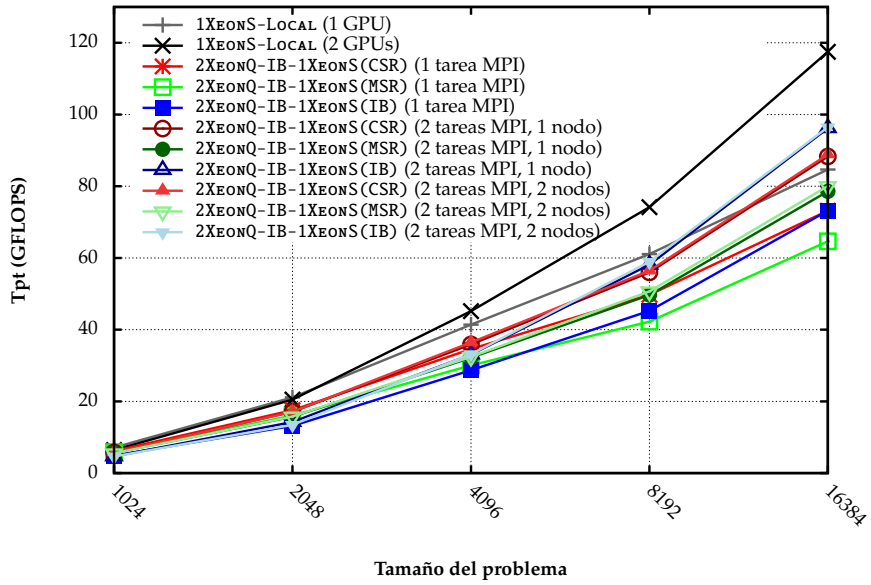
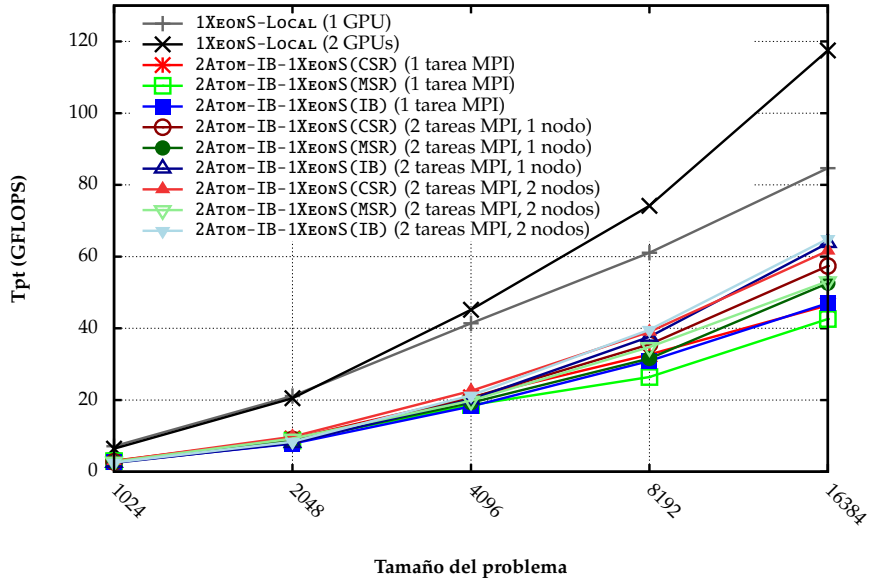


Figura 4.13: Comparativa del rendimiento (Tpt) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red 1GbE frente a las ejecuciones aceleradas con las GPUs locales.



(a) Cliente XeonQ



(b) Cliente Atom

Figura 4.14: Comparativa del rendimiento (Tpt) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red IB frente a las ejecuciones aceleradas con las GPUs locales.

procesadores Intel Xeon que incorporan las plataformas XeonQ disponen de una capacidad de cálculo superior a los procesadores Intel Atom de las máquinas Atom, de modo que ejecutan más rápidamente la porción del problema que el benchmark asigna a la CPU.

Ahora bien, con independencia de la plataforma utilizada como cliente, las configuraciones con tecnologías 10GbE (Figuras 4.11 y 4.12) obtienen su mayor rendimiento computacional utilizando el módulo de comunicaciones CSR. Como muestra la Figura 4.14, esta situación se repite en las configuraciones con tecnología IB y tamaños de problema menores a 16.384. Sin embargo, en los casos en que $n = 16.384$, el módulo IB permite que las ejecuciones del benchmark finalicen más rápidamente.

Claramente, esta pérdida de rendimiento y aumento de consumo del módulo CSR frente a IB para tamaños de problema $n = 16.384$ se debe a la longitud de los datos transferidos entre el cliente y el servidor rCUDA. En la Sección 4.2 se ha demostrado que el ancho de banda obtenido por el módulo CSR es inferior al de IB únicamente cuando el tamaño de los datos transferidos es mayor a 1 MB.

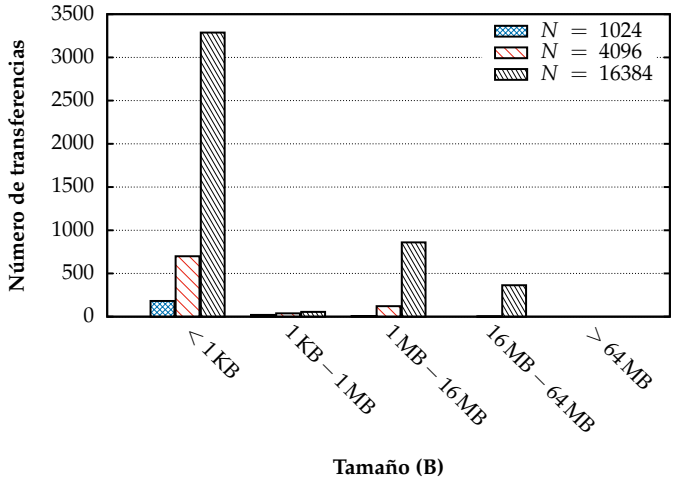


Figura 4.15: Estudio del tamaño de las transferencias efectuadas por rCUDA durante la ejecución de HPL-Fermi para una malla con 1 tarea MPI.

Con el objetivo de confirmar que el tamaño de las transferencias es la causa de este comportamiento, se ha configurado el software de rCUDA para que mantenga un registro de los tamaños de todas las transferencias

que se llevan a cabo entre el cliente y el servidor. De este modo se ha recopilado información del tamaño de las transferencias efectuadas en las ejecuciones del benchmark con un tamaño de problema 1.024 , 4.096 y 16.384. Los resultados, representados en la Figura 4.15, indican que para $n = 1.024$ se producen 28 (un 13,20 %) transferencias de tamaño mayor a 1 MB, mientras que para $n = 16.384$ se han ejecutado 1.280 (un 28,02 %). Esto demuestra que al aumentar el tamaño del problema se incrementa la longitud de los datos transferidos, y por tanto el módulo IB mejora el rendimiento de CSR.

Respecto a la evolución del *speed-up* relativo a la ejecución acelerada con 1 GPU local, la Tabla 4.7 muestra que al incrementar el tamaño del problema el factor de mejora del rendimiento se mantiene constante o incluso se incrementa. Así, por ejemplo, el *speed-up* de 2XeonQ-IB-1XeonS al utilizar el módulo IB para $n = 1.024$ es 0,69 $\times$, mientras que cuando $n = 16.384$ es 0,86 $\times$. Por otro lado, al emplear el módulo CSR se obtiene una aceleración 0,86 $\times$ para ambos tamaños.

Escenario	Tamaño del problema				
	1.024	**2.048**	**4.096**	**8.192**	**16.384**
2XeonQ-IB-1XeonS(CSR)	0,86	0,83	0,84	0,81	0,86
2XeonQ-IB-1XeonS(MSR)	0,82	0,76	0,73	0,69	0,76
2XeonQ-IB-1XeonS(IB)	0,69	0,62	0,69	0,74	0,86
2Atom-IB-1XeonS(CSR)	0,43	0,44	0,50	0,54	0,55
2Atom-IB-1XeonS(MSR)	0,42	0,41	0,45	0,43	0,50
2Atom-IB-1XeonS(IB)	0,37	0,37	0,44	0,51	0,56
1XeonQ-RoCE-1XeonS(CSR)	0,80	0,70	0,70	0,71	0,78
1XeonQ-RoCE-1XeonS(MSR)	0,76	0,62	0,55	0,59	0,61
1Atom-RoCE-1XeonS(CSR)	0,42	0,41	0,46	0,50	0,53
1Atom-RoCE-1XeonS(MSR)	0,40	0,37	0,38	0,39	0,38
1XeonQ-iWARP-1XeonS(CSR)	0,79	0,70	0,71	0,71	0,78
1XeonQ-iWARP-1XeonS(MSR)	0,74	0,61	0,55	0,59	0,61
2XeonQ-1GbE-1XeonS(TCP)	0,37	0,20	0,17	0,18	0,16
2Atom-1GbE-1XeonS(TCP)	0,25	0,16	0,14	0,15	0,14

Tabla 4.7: Factor de mejora del rendimiento (*speed-up*) de las configuraciones con virtualización de GPUs al resolver el sistema de ecuaciones de HPL-Fermi empleando una malla de 1 tarea MPI (relativo a la ejecución acelerada con una GPU local sobre 1XeonS-Local).

4.5.2. Aceleración con dos GPUs

Para evaluar el rendimiento de las configuraciones ante el escalado de HPL-Fermi se ha ejecutado el benchmark con una distribución ($q \times p$) de 1×2 tareas MPI (dos tareas MPI acceden a dos GPUs remotas), donde las dos tareas se ubican en el mismo nodo y, si es posible, en dos nodos diferentes de las configuraciones. De nuevo, para poner en contexto los resultados obtenidos, se muestran los valores de rendimiento y energía de la ejecución acelerada con las 2 GPUs locales de 1XeonS-Local

Al examinar las Tablas 4.6, 4.8 y 4.9, se advierte que el escalado del benchmark permite a las configuraciones disminuir el tiempo de ejecución obtenido en las pruebas con único proceso, mejorando incluso el tiempo de la ejecución acelerada con 1 GPU local.

Concretamente, las ejecuciones que distribuyen las tareas MPI entre dos nodos diferentes proporcionan mayor rendimiento en las configuraciones con tecnología IB. La Figura 4.14, muestra que, al igual que ocurre en las ejecuciones con una malla de un único proceso, los escenarios que utilizan el módulo CSR suministran mejores prestaciones con los tamaños de problema más pequeños. Así, para las ejecuciones con $n = 1.024$ en 2XeonQ-IB-1XeonS, utilizando CSR se tiene una duración 1,25 $\times$ veces menor que con el módulo IB, y 1,08 $\times$ que MSR. Por otro lado, en las ejecuciones con tamaños de problema grandes se consigue un mayor rendimiento empleando el módulo IB. Esto es nuevamente consecuencia del tamaño de las transferencias efectuadas por el benchmark.

Contrariamente, en los escenarios con redes 1GbE, las pruebas donde los procesos se ejecutan sobre el mismo nodo son más rápidas. Es evidente que el sistema de interconexión se convierte en un cuello de botella en este tipo de configuraciones, por eso se obtienen mejores prestaciones en los escenarios donde las tareas no efectúan transferencias entre sí a través de la red 1GbE. Por esta razón, al ubicar las 2 tareas sobre un mismo nodo en 2XeonQ-1GbE-1XeonS, la ejecución finaliza en 216,1 s, que se corresponde con el 85 % del tiempo obtenido al separar los procesos en los dos nodos XeonQ.

Como se aprecia claramente en las Figuras 4.11 y 4.12 el módulo CSR sigue siendo la mejor opción para alcanzar el rendimiento máximo sobre las configuraciones con redes 10GbE. Por ejemplo, las pruebas para un ta-

maño de $n = 16.384$ sobre 1XeonQ-iWARP-1XeonS y 1XeonQ-RoCE-1XeonS empleando el módulo CSR obtienen un *speed-up* de 0,78 × por 0,61 × de MSR.

Escenario	Tamaño del problema									
	1.024		2.048		4.096		8.192		16.384	
	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)
1XeonS-Local	0,11	14	0,28	38	1,01	147	4,94	808	25,0	4.677
2XeonQ-IB-1XeonS(CSR)	0,12	18	0,34	56	1,28	224	6,55	1.202	33,2	6.997
2XeonQ-IB-1XeonS(MSR)	0,13	19	0,37	60	1,42	239	7,38	1.274	37,3	7.313
2XeonQ-IB-1XeonS(IB)	0,15	20	0,40	59	1,39	217	6,31	1.082	30,5	6.022
2Atom-IB-1XeonS(CSR)	0,26	32	0,64	82	2,23	303	10,32	1.504	51,1	8.170
2Atom-IB-1XeonS(MSR)	0,27	32	0,67	83	2,38	319	11,61	1.598	55,8	8.491
2Atom-IB-1XeonS(IB)	0,29	36	0,72	87	2,27	291	9,74	1.330	46,0	7.170
1XeonQ-RoCE-1XeonS(CSR)	0,13	20	0,40	66	1,59	274	7,54	1.415	36,9	7.663
1XeonQ-RoCE-1XeonS(MSR)	0,14	21	0,44	67	1,84	289	9,17	1.461	46,0	8.401
1Atom-RoCE-1XeonS(CSR)	0,27	34	0,69	88	2,43	322	11,08	1.561	53,5	8.392
1Atom-RoCE-1XeonS(MSR)	0,28	32	0,75	89	2,78	357	13,96	1.770	67,3	9.807
1XeonQ-iWARP-1XeonS(CSR)	0,13	20	0,45	70	1,63	285	8,28	1.424	38,9	7.855
1XeonQ-iWARP-1XeonS(MSR)	0,14	22	0,45	72	1,85	303	9,23	1.540	48,3	8.881
2XeonQ-1GbE-1XeonS(TCP)	0,30	43	1,46	217	7,11	1.100	34,44	5.533	190,2	32.392
2Atom-1GbE-1XeonS(TCP)	0,46	57	2,02	261	9,01	1.242	40,72	5.824	214,8	31.887

Tabla 4.8: Tiempo (TtS) y energía (EtS) requeridos por un nodo para resolver el sistema de ecuaciones de HPL-Fermi utilizando 2 tareas MPI.

Escenario	Tamaño del problema									
	1.024		2.048		4.096		8.192		16.384	
	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)	TtS (s)	EtS (J)
1XeonS-Local	0,11	14	0,28	38	1,01	147	4,94	808	25,0	4.677
2XeonQ-IB-1XeonS(CSR)	0,12	21	0,34	61	1,26	243	6,49	1.290	33,0	7.531
2XeonQ-IB-1XeonS(MSR)	0,13	21	0,36	62	1,41	251	7,23	1.348	36,6	7.573
2XeonQ-IB-1XeonS(IB)	0,16	25	0,43	68	1,38	240	6,22	1.134	30,4	6.394
2Atom-IB-1XeonS(CSR)	0,25	32	0,59	77	2,03	279	9,40	1.357	47,6	7.828
2Atom-IB-1XeonS(MSR)	0,25	31	0,62	78	2,31	291	10,55	1.471	54,8	8.649
2Atom-IB-1XeonS(IB)	0,28	34	0,69	85	2,15	279	9,26	1.313	45,1	7.110
2XeonQ-1GbE-1XeonS(TCP)	0,47	72	1,92	294	8,81	1.285	40,59	6.928	221,4	40.047
2Atom-1GbE-1XeonS(TCP)	0,65	88	2,57	351	10,90	1.480	46,42	6.754	245,5	36.476

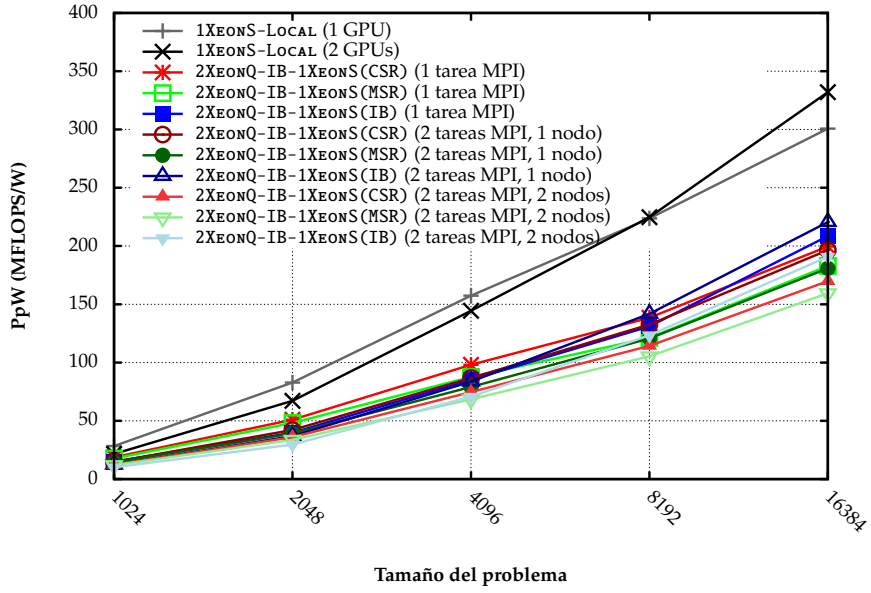
Tabla 4.9: Tiempo (TtS) y energía (EtS) requeridos por dos nodos para resolver el sistema de ecuaciones de HPL-Fermi utilizando 2 tareas MPI.

4.5.3. Consumo energético de las configuraciones con GPUs remotas

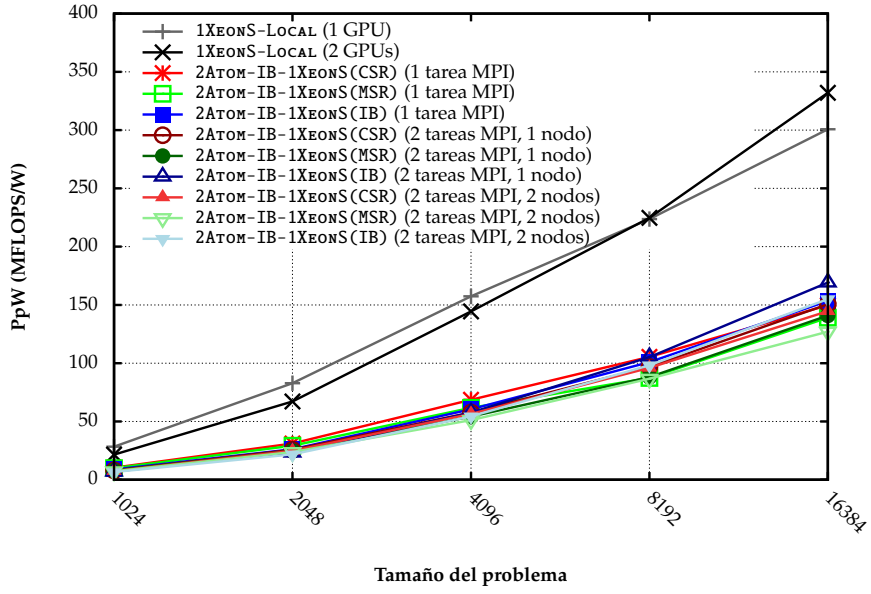
Desde el punto de vista del rendimiento energético, es obvio que la utilización de dos nodos supone un aumento de la potencia eléctrica consumida y una disminución del rendimiento por vatio, como se observa en la Figura 4.16. Además, las Figuras 4.16, 4.17 y 4.18 también muestran que la ejecución de dos tareas MPI de HPL-Fermi, aceleradas mediante GPUs remotas sobre redes RDMA, para tamaños de problema pequeños (1.024, 2.048 y 4.096) es menos eficiente en términos energéticos que ejecutar una sola tarea. Sin embargo, como se ha indicado anteriormente, 2 tareas MPI de HPL-Fermi consiguen un tiempo de ejecución menor que las ejecuciones con una sola tarea.

Por otro lado, los resultados de las Tablas 4.6, 4.8 y 4.9 revelan que las configuraciones con plataformas XeonQ presentan un consumo energético menor que las configuraciones con las máquinas Atom, siempre y cuando la relación entre los tiempos de ejecución de Atom y XeonQ sea menor que la relación entre las potencias medias de XeonQ y Atom durante la ejecución de la aplicación. Esta situación se aprecia claramente en las ejecuciones con tamaños de problema 8.192 y 16.384 sobre las configuraciones con red 1GbE, donde el cuello de botella generado por la red hace que el factor de mejora del tiempo de ejecución de XeonQ se reduzca. Por ejemplo, la ejecución para $n = 16.384$, con una distribución de dos procesos sobre un nodo en 2XeonQ-1GbE-1XeonS, es 24 s más corta y consume 505 J más que en 2Atom-1GbE-1XeonS. En este caso, la relación entre los tiempos de ejecución es 1,13 $\times$, mientras que la relación entre la potencia media de ambas configuraciones es 1,15 $\times$.

Para finalizar, con el objeto de poner en contexto la eficiencia energética alcanzada por las configuraciones con virtualización, se calcula la posición que ocuparían dentro de la lista Green500 y Little500 mantenida en [15]. La Tabla 4.10 muestra la clasificación de cada una de las configuraciones utilizadas en los experimentos frente a los computadores más eficientes utilizados actualmente en HPC, considerando como índice de referencia el mejor PpW presentado para un tamaño de problema $n = 16.384$.

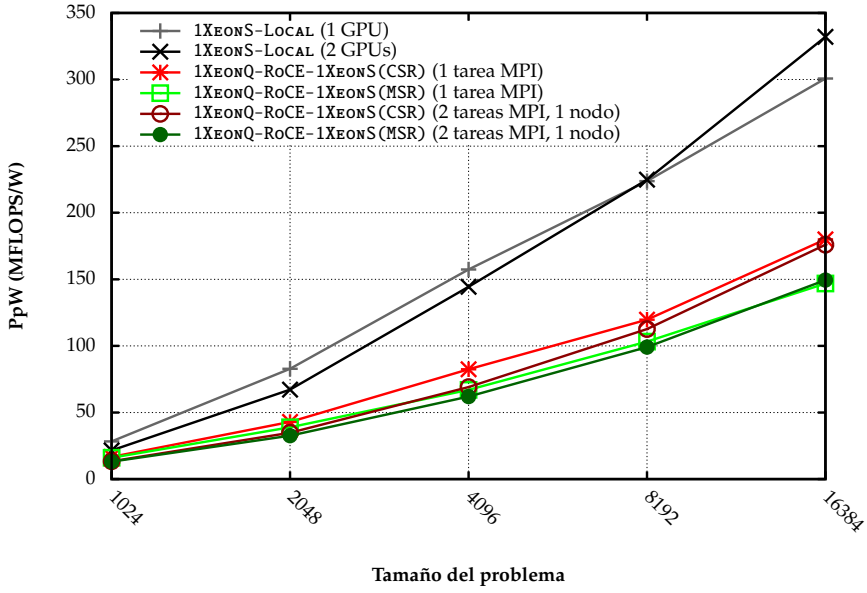


(a) Cliente XeonQ

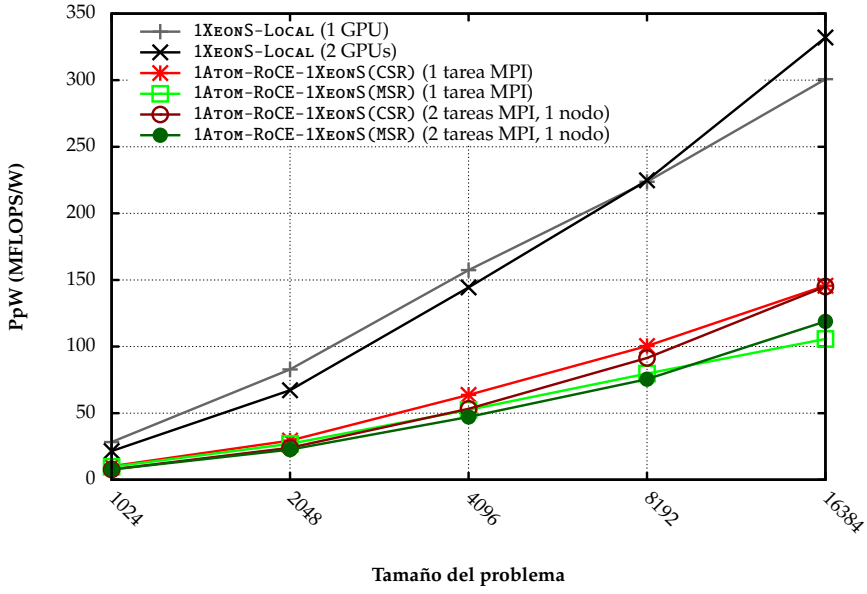


(b) Cliente Atom

Figura 4.16: Comparativa del rendimiento por vatio (PpW) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red IB frente a las ejecuciones aceleradas con las GPUs locales.



(a) Cliente XeonQ



(b) Cliente Atom

Figura 4.17: Comparativa del rendimiento por vatio (PpW) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red RoCE frente a las ejecuciones aceleradas con las GPUs locales.

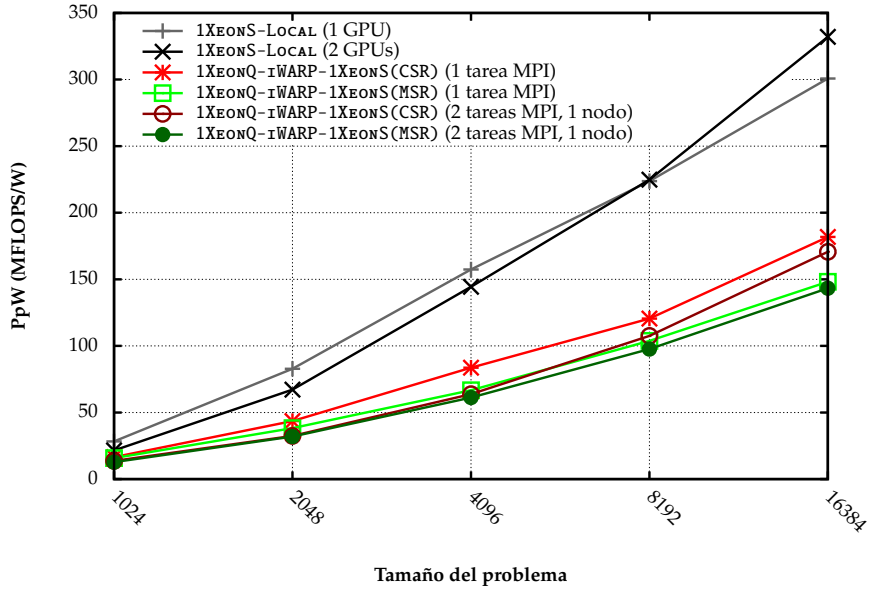


Figura 4.18: Comparativa del rendimiento por vatio (PpW) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red iWARP frente a las ejecuciones aceleradas con las GPUs locales.

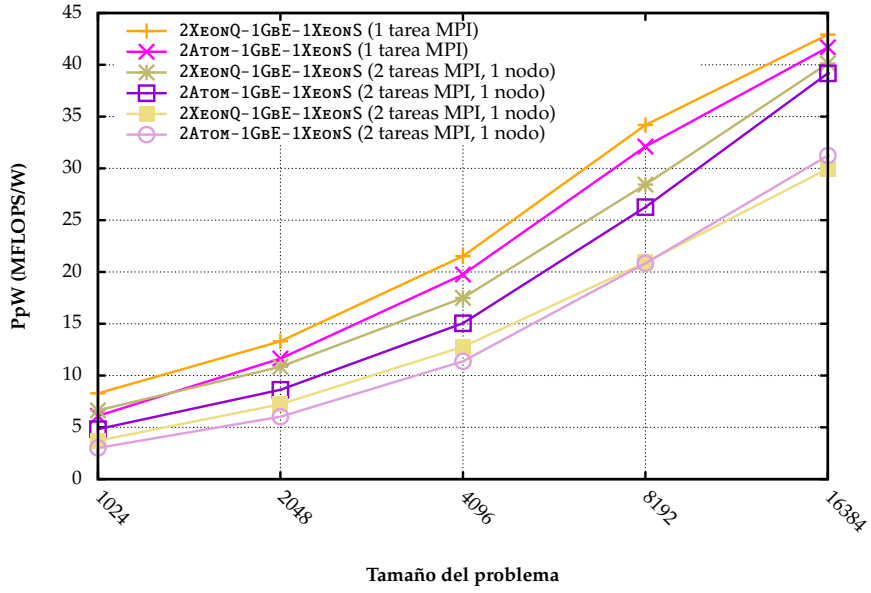


Figura 4.19: Comparativa del rendimiento por vatio (PpW) de las ejecuciones de HPL-Fermi aceleradas mediante GPUs remotas a través de una red 1GbE frente a las ejecuciones aceleradas con las GPUs locales.

Configuración	Módulo	PpW (MFLOPS/W)	Posición Green500 (Jun. 2015)	Posición Little500 (Nov. 2014)
1XeonS-Local		332,06	311	189
2XeonQ-IB-1XeonS	IB	220,54	372	264
2Atom-IB-1XeonS	IB	169,16	426	401
1XeonQ-RoCE-1XeonS	CSR	179,98	422	370
1Atom-RoCE-1XeonS	CSR	145,69	440	405
1XeonQ-rWARP-1XeonS	CSR	181,86	417	341
2XeonQ-1GbE-1XeonS	TCP	42,92	501	501
2Atom-1GbE-1XeonS	TCP	41,68	501	501

Tabla 4.10: Clasificación de las configuraciones utilizadas en la listas Green500 y Little500 [15].

4.5.4. Conclusiones

Hay que indicar que, aún siendo siempre mejores los resultados sin virtualización, los valores obtenidos en la evaluación de HPL-Fermi han demostrado la viabilidad de una agrupación o *pool* de GPUs remotas, de manera que rCUDA hace que cada uno de los nodos de un clúster tenga a su disposición todas las GPUs del conjunto, eliminando así la limitación en la asignación de recursos y aumentando su utilización.

Por otro lado, se ha observado que los módulos CSR y MSR diseñados en el presente trabajo permiten aumentar el rendimiento y reducir el consumo de las configuraciones con rCUDA al ejecutar un benchmark basado en la resolución de un problema matemático.

Además, los resultados indican que la utilización de las plataformas de bajo consumo Atom, junto con la virtualización de GPUs, pueden proporcionar un ahorro energético significativo frente a XeonQ cuando el tiempo de ejecución es elevado, a cambio de introducir un ligero sobrecoste temporal.

Finalmente, se debe resaltar que, del conjunto de configuraciones con plataformas de bajo consumo, únicamente se han podido evaluar aquellas que utilizan Atom, ya que la arquitectura ARM no está soportada por la aplicación.

4.6. Evaluación de LAMMPS

LAMMPS (*Large-scale Atomic/Molecular Massively Parallel Simulator*) es una aplicación de dinámica molecular clásica que modela conjuntos de partículas en estado líquido, sólido o gaseoso. Puede simular sistemas de partículas atómicas, de polímeros, biológicas o metálicas sometidas a campos de fuerza y condiciones de entorno. Cuenta con más de 5000 publicaciones [129]. Su importancia y su expansión se debe a que es una aplicación de código abierto capaz de representar casi cualquier tipo de modelo de partículas o campo de fuerza conocido. Además, en los últimos años, LAMMPS se ha convertido en una de las aplicaciones de producción más habituales en las pruebas de rendimiento y en los análisis comparativos de las tecnologías de HPC [52, 130, 131].

LAMMPS puede ejecutarse sobre sistemas de memoria distribuida utilizando una API de MPI. Esto le permite emplear una técnica de descomposición espacial para dividir el dominio de simulación en pequeños subdominios que se reparten entre las tareas MPI de una malla tridimensional.

Otra de las características de LAMMPS es que incorpora módulos software optimizados para ciertos tipos de hardware, incluyendo varias CPUs multinúcleo, aceleradores gráficos y coprocesadores Intel Xeon Phi. En particular, LAMMPS dispone de dos módulos o paquetes basados en CUDA: `USER-CUDA` y `GPU`. Ambos permiten redirigir ciertos cálculos a las GPUs locales para mejorar su rendimiento. Sin embargo, cada uno de ellos ofrece unos beneficios en función de la simulación efectuada.

En la experimentación se ha empleado la versión de LAMMPS publicada el 1 de febrero de 2014. El ejecutable ha sido enlazado con las bibliotecas OpenMPI, CUDA y FFTW [132] disponibles en las plataformas de evaluación.

Asimismo, se ha empleado el módulo `USER-CUDA` para acceder a la aceleración facilitada por CUDA. Este módulo establece una configuración paralela híbrida para LAMMPS donde cada tarea MPI utiliza un núcleo del procesador por GPU disponible. Con ello se ha pretendido reducir la aportación de la CPU al rendimiento final obtenido.

La carga de prueba ha sido generada mediante la simulación de fluidos Lennard-Jones (`lj`) incluida en el paquete de distribución del código fuente

de LAMMPS. La simulación se ha realizado utilizando 1, 2, 4 o 8 tareas MPI (con una sola hebra), distribuidas en 1 o 2 nodos, donde cada tarea tiene acceso a 1 GPU. El test ha sido configurado para efectuar 2.000 iteraciones con un tamaño de problema de 64, lo que genera un conjunto de cerca de $1{,}05 \cdot 10^6$ átomos. Este tamaño de problema ha sido escogido con el objetivo de que las GPUs de las plataformas de experimentación dispongan de memoria suficiente para almacenar los datos de un máximo de 4 tareas MPI simultáneas.

Los resultados de rendimiento presentados a continuación han sido facilitados por la aplicación durante la ejecución de las pruebas. Se corresponden con el tiempo requerido para completar la simulación en los escenarios con GPUs locales y con servidores rCUDA en XeonS. Las pruebas preliminares han demostrado que el resto de escenarios (con clientes en plataformas de 64 bits y servidores en máquinas de 32 bits) no son funcionales, ya que la distribución de LAMMPS para arquitecturas Intel de 32 bits no soporta plataformas hardware Intel de 64 bits.

4.6.1. Aceleración mediante GPUs locales

En este apartado se evalúa la ejecución de LAMMPS cuando se acelera utilizando las GPUs locales de las configuraciones de la Tabla 4.4.

A la izquierda de la gráfica en la Figura 4.20 se observa que los tiempos de ejecución obtenidos al utilizar una única tarea MPI en las configuraciones 1XeonS-Local y 1Kayla-Local están próximos (92,24 s y 120,08 s, respectivamente). Ambas mejoran el rendimiento proporcionado por la configuración 2Carma-Local en 8,04 × y 5,81 ×, y por 2Jetson-Local en 14,03 × y 10,18 ×. En cambio, cuando el número de tareas MPI utilizadas por la aplicación aumenta, el tiempo de ejecución de 1Kayla-Local se distancia del obtenido en 1XeonS-Local. En concreto, LAMMPS requiere un 321,14 % más de tiempo al utilizar 2 tareas MPI en 1Kayla-Local y únicamente un 13,56 % más en 1XeonS-Local frente a sus respectivas ejecuciones con 1 tarea MPI. Estos resultados indican que esta aplicación realiza un uso intensivo de la GPU. De ahí que 1Kayla-Local, al disponer de una GPU con mayor capacidad de cálculo, obtenga mejores resultados que las otras configuraciones con plataformas ARM. Sin embargo, todas las configuraciones con plataformas ARM sufren una penalización similar

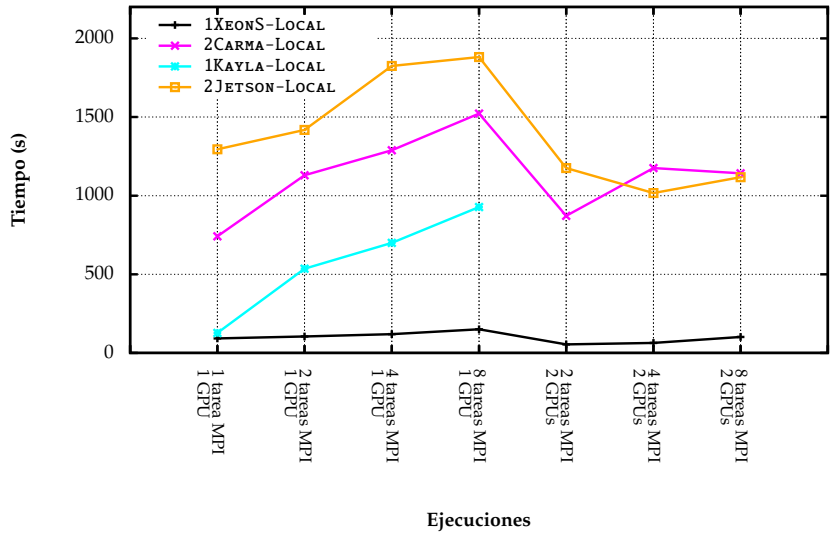


Figura 4.20: Tiempo de ejecución (TtS) de las configuraciones sin virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs locales.

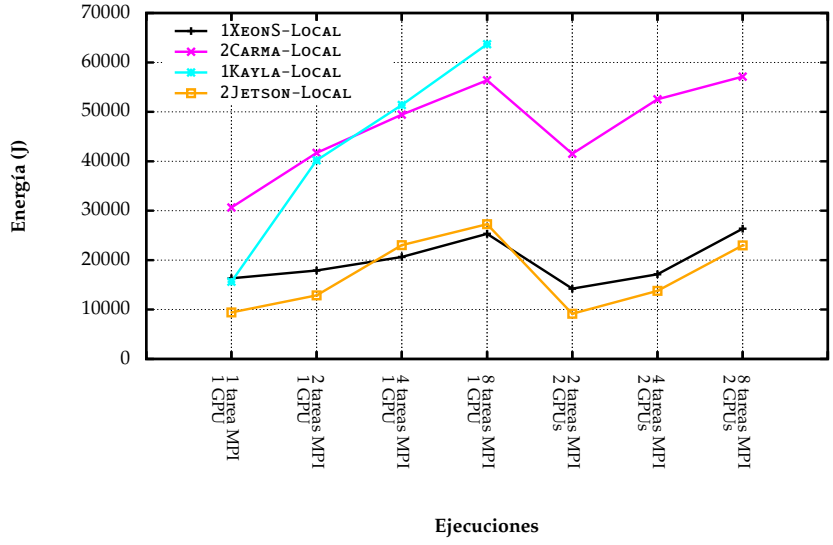


Figura 4.21: Energía (EtS) requerida por las configuraciones sin virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs locales.

cuando se incrementa el número de tareas MPI que utiliza la aplicación, cosa que no ocurre en 1XeonS-Local al disponer éste de un bus PCIe con mayor ancho de banda.

No obstante, el uso intensivo de la GPU no justifica los pobres resultados obtenidos por 2Jetson-Local frente a 2Carma-Local al emplear un nodo, sobre todo si se tiene en consideración que los rendimientos de las respectivas GPUs son 300 GFLOPS [133] y 270 GFLOPS [134]. Para determinar la causa es necesario realizar un análisis de las transferencias que efectúa la aplicación entre la CPU y la GPU. Con este propósito se ha configurado nuevamente el software de rCUDA para realizar un informe de los tamaños de las transferencias efectuadas entre el cliente y el servidor al ejecutar 1 y 2 tareas MPI de LAMMPS aceleradas con 1 GPU remota.

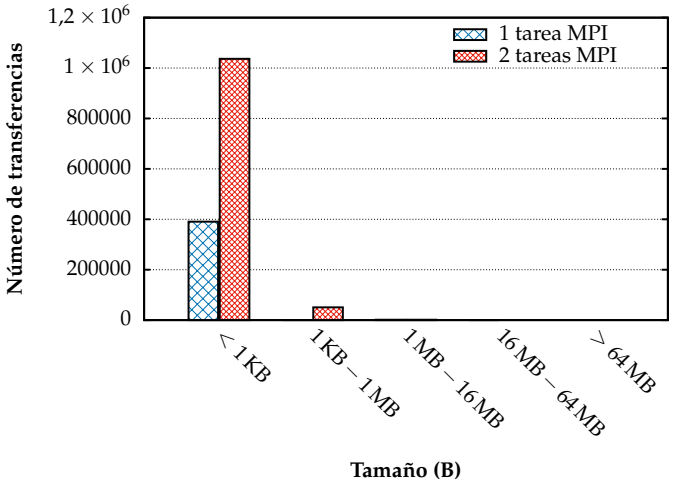


Figura 4.22: Estudio del tamaño de las transferencias efectuadas por rCUDA durante la ejecución de LAMMPS utilizando 1 y 2 tareas MPI cooperativas.

En la Figura 4.22, se aprecia el elevado número de transferencias de datos con tamaño inferior a 1 KB que realiza LAMMPS. Concretamente, cuando la aplicación utiliza 1 tarea MPI efectúa 390.900 transferencias menores de 1 KB, que se corresponde con el 99,58 % del total; mientras que al disponer de 2 tareas MPI se incrementa hasta 1.037.125 , lo que supone el 95,23 %. Como se ha mostrado en la Sección 4.2, las plataformas Jetson presentan un ancho de banda muy bajo para transferencias entre la CPU y la GPU con un tamaño menor a 50 KB. Esto penaliza las ejecuciones de

LAMMPS en la configuración 2Jetson-Local frente al resto.

Por otro lado, en la Sección 4.2 también se ha constatado una limitación en el rendimiento de las transferencias sobre infraestructuras 1GbE para los interfaces de red de las máquinas Carma y Kayla. La Figura 4.20 revela la influencia de este problema en las ejecuciones aceleradas con las 2 GPUs locales de la configuración 2Carma-Local, ya que en estos casos la diferencia de rendimiento con 2Jetson-Local se reduce, o incluso se invierte. Así, por ejemplo, cuando LAMMPS utiliza 4 tareas MPI y 2 GPUs tarda 1.016 s en 2Jetson-Local frente a 1.176 s en 2Carma-Local.

Escenario	Número de tareas MPI						
	1 GPU				2 GPUs		
	1	2	4	8	2	4	8
1XeonS-Local	343	337	340	335	430	435	427
2Carma-Local	47	42	44	48	58	55	61
2Jetson-Local	14	16	20	28	22	28	35
1Kayla-Local	197	149	148	143	–	–	–

Tabla 4.11: Potencia media (HLP) requerida por las configuraciones sin virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs locales (en W).

Desde la perspectiva energética, es visible en la Figura 4.21 que las configuraciones 2Jetson-Local y 1XeonS-Local presentan el menor consumo con independencia del número de tareas MPI y su distribución. En concreto, el consumo de energía de las ejecuciones con 8 tareas MPI compartiendo 1 GPU en 1XeonS-Local es 25.330 J, que se corresponde con el 92,85 % del realizado en 2Jetson-Local, el 44,90 % de 2Carma-Local y el 39,77 % de 1Kayla-Local.

No obstante, a pesar de que ambos ofrecen un consumo similar, su potencia eléctrica media no es la misma. En particular, analizando la Tabla 4.11, se observa que 1XeonS-Local es la configuración que más potencia requiere, mientras que 2Jetson-Local es la que menos precisa.

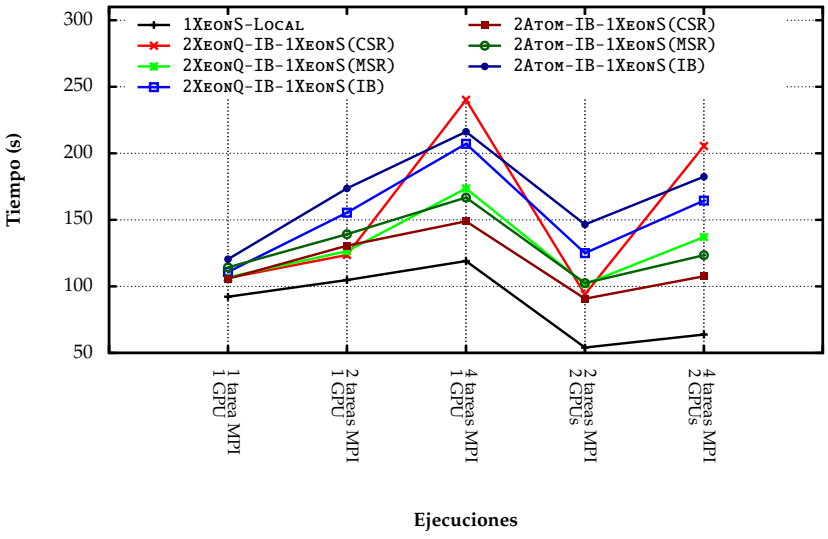
Respecto a las configuraciones 2Carma-Local y 1Kayla-Local, la Figura 4.21 indica que ambas presentan un consumo muy elevado. En el caso de 2Carma-Local, la causa es su pésimo tiempo de ejecución unido a una potencia media situada entre 42 W y 61 W. Por otro lado, en 1Kayla-Local es debido principalmente a su considerable potencia media, siendo en todos los casos mayor que 140 W.

4.6.2. Aceleración mediante GPUs remotas

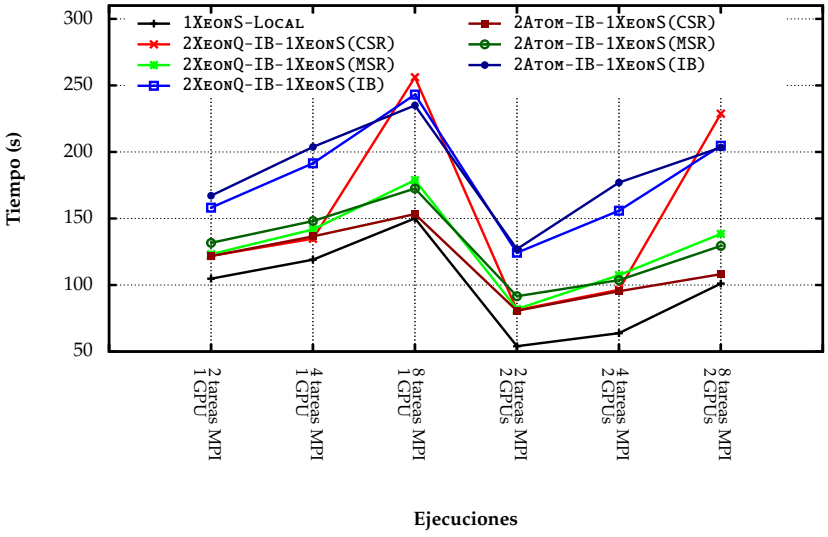
En este apartado se evalúan las prestaciones proporcionadas por las configuraciones descritas en las Tablas 4.2 y 4.3, de forma que diferentes tareas MPI de LAMMPS ubicadas en un mismo nodo solicitan los servicios de 1 o 2 GPUs remotas utilizando el software intermediario de rCUDA. Además, en aquellas configuraciones que es posible, las ejecuciones se han repetido distribuyendo las tareas MPI entre los 2 nodos de la configuración.

Desde el punto de vista de tiempo de ejecución, en las Figuras 4.23, 4.24 y 4.25 se observa que las configuraciones con tecnología RDMA finalizan más rápidamente la simulación utilizando el módulo CSR cuando la cantidad de tareas MPI de LAMMPS por nodo es menor que el número de núcleos. Así, por ejemplo, la configuración `2XeonQ-IB-1XeonS` realiza la simulación con 1 tarea MPI acelerada con 1 GPU remota utilizando CSR en 106,35 s frente a 107,84 s para MSR y 110,79 s para IB. En cambio, si el número de tareas MPI de LAMMPS asignadas a un nodo es igual o superior al número de núcleos del procesador, el rendimiento del módulo CSR se degrada y ofrece el peor resultado. Entonces, el mejor tiempo de ejecución se alcanza con el módulo MSR. Esto se aprecia claramente en las configuraciones donde se distribuyen 4 tareas MPI sobre una misma máquina `XeonQ`. Por ejemplo, la duración de la ejecución acelerada con 2 GPUs remotas utilizando el módulo CSR en `2XeonQ-RoCE-1XeonS` es 1,41 $\times$ veces más larga que con MSR. En estos casos, las operaciones de cálculo ejecutadas por las tareas MPI de la aplicación junto con la carga generada por las operaciones de comunicación de la semántica de canal del módulo CSR saturan el sistema y reducen sus prestaciones.

Dejando al margen la situación anterior, la Figura 4.23 muestra que las configuraciones con infraestructuras de red IB aumentan su tiempo de ejecución cuando utilizan el módulo IB. Tal es el caso de `2XeonQ-IB-1XeonS`, donde las ejecuciones con 2 tareas MPI aceleradas con 2 GPUs sobre los 2 nodos empleando el módulo IB consumen 124,25 s, mientras que CSR y MSR finalizan en 81,15 s y 82,09 s, respectivamente. Esto es consecuencia de que la mayor parte de las transferencias de LAMMPS son de pequeño tamaño (Figura 4.22), y como se ha demostrado en la Sección 4.2, las transferencias de tamaño menor 2 MB penalizan el rendimiento del módulo IB frente a los otros dos.



(a) Sobre un único nodo cliente



(b) Sobre dos nodos cliente

Figura 4.23: Tiempo de ejecución (TtS) de las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red IB.

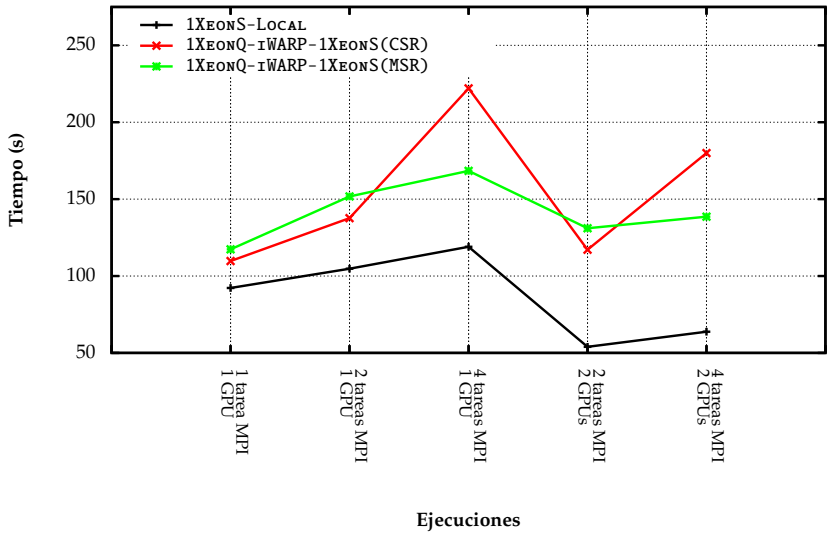


Figura 4.24: Tiempo de ejecución (TtS) de las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red iWARP.

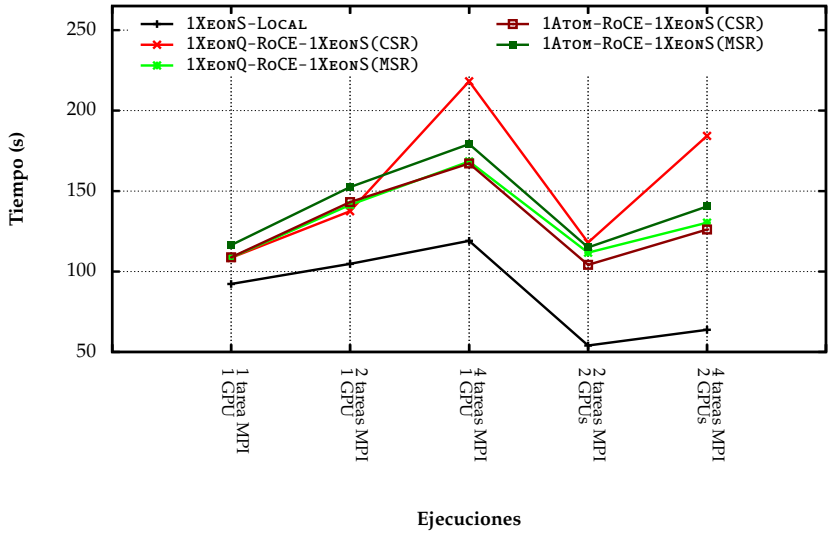
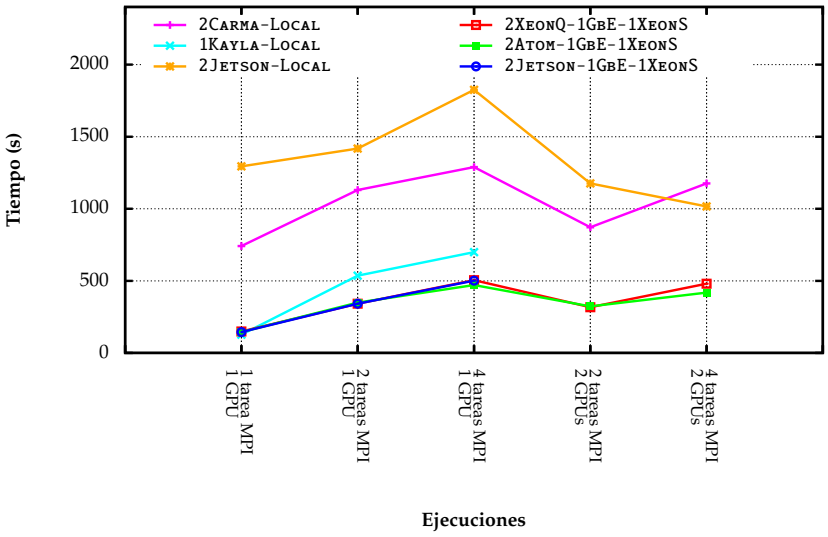
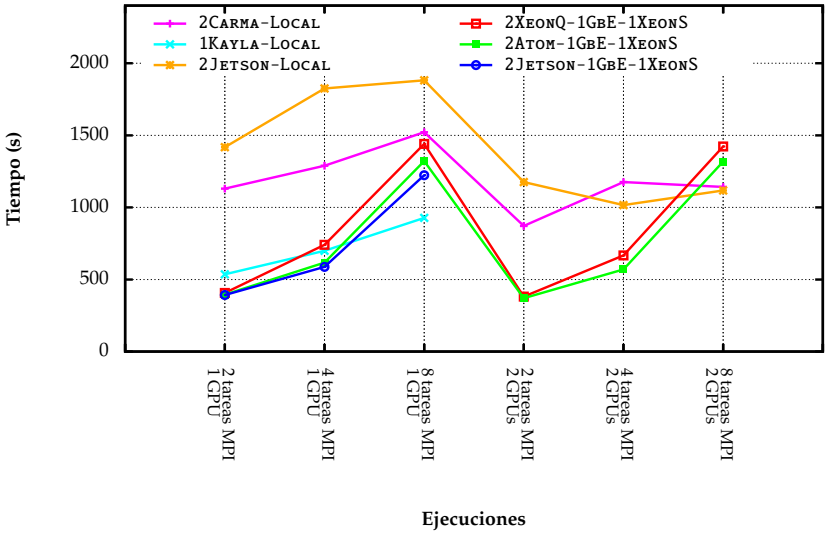


Figura 4.25: Tiempo de ejecución (TtS) de las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red RoCE.



(a) Sobre un único nodo cliente



(b) Sobre dos nodos cliente

Figura 4.26: Tiempo de ejecución (TtS) de las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red 1GbE.

Por otro lado, el módulo USER-CUDA de LAMMPS realiza la mayor parte de los cálculos en la GPU. De ahí que las ejecuciones que distribuyen más de una tarea MPI por GPU no proporcionen un incremento del rendimiento. Más aún, incluso se produce una reducción del rendimiento a causa de la sobrecarga generada por las comunicaciones y los eventos de sincronización de las tareas, como se aprecia en la Figura 4.20. Las Figuras 4.23, 4.24 y 4.25 confirman un comportamiento similar en las configuraciones con virtualización. Sin embargo, si se considera la mejor configuración en términos de tiempo de ejecución y estabilidad, que se corresponde con 2Atom-IB-1XeonS(CSR), y se compara con los resultados obtenidos sobre 1XeonS-Local con 1 y 2 GPUs locales (Figura 4.27), se observa que el sobrecoste introducido por el acceso remoto disminuye. En particular, cuando aumenta el número de tareas MPI, este sobrecoste se desciende hasta un 1,67 % al utilizar 1 GPU y un 7,05 % con 2 GPUs, donde la desviación relativa en ambos casos es menor al 3,5 %. Esto confirma que las configuraciones con rCUDA responden de forma más eficiente ante las aplicaciones que utilizan tareas cooperativas estrechamente sincronizadas que realizan las comunicaciones y los cálculos al mismo tiempo.

En cuanto a las configuraciones con infraestructura de red 1GbE, en la Figura 4.26 se puede ver que, salvo en el caso de las ejecuciones con 8 tareas MPI, las configuraciones con virtualización de GPUs proporcionan un tiempo de ejecución menor que las configuraciones basadas en arquitecturas ARM que realizan la aceleración mediante GPUs locales. En concreto, utilizando el sistema Jetson como cliente y XeonS como servidor se obtiene el mejor resultado (para 8 tareas MPI y 1 GPU 1.222,90 s, comparados con 1.321,57 s y 1.440,89 s para las combinaciones con Atom y XeonQ como clientes, respectivamente) cuya desviación relativa máxima es del 2,6 %.

Otro rasgo destacable de las configuraciones con 1GbE es que, al incrementar el número de GPUs, no se aumenta el rendimiento como ocurre con las configuraciones con RDMA. Es evidente que el sobrecoste que penaliza las configuraciones con red 1GbE surge principalmente de la utilización de una red de interconexión lenta.

Respecto al consumo energético, al tener tiempos de ejecución tan similares, las configuraciones donde los clientes son plataformas de bajo consumo (Atom y Jetson) presentan un ahorro energético frente a las configuraciones con equipos XeonQ. Por ejemplo, en la Figura 4.31 se observa

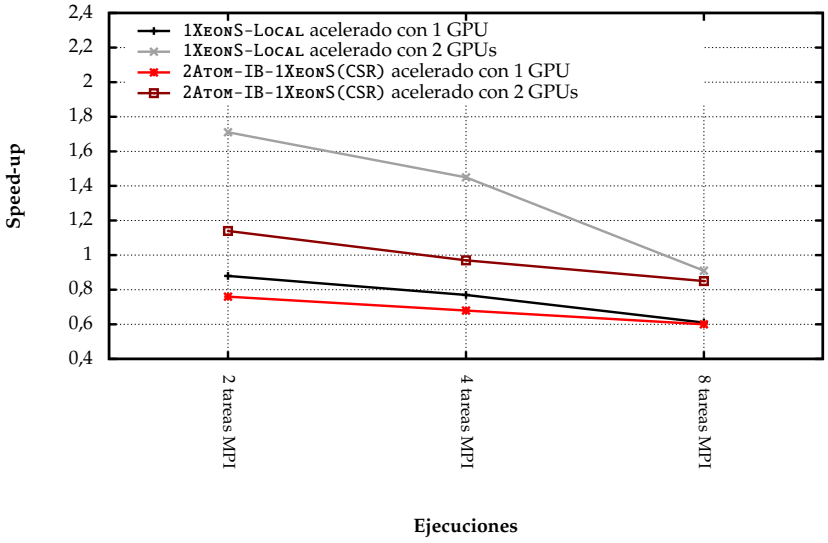


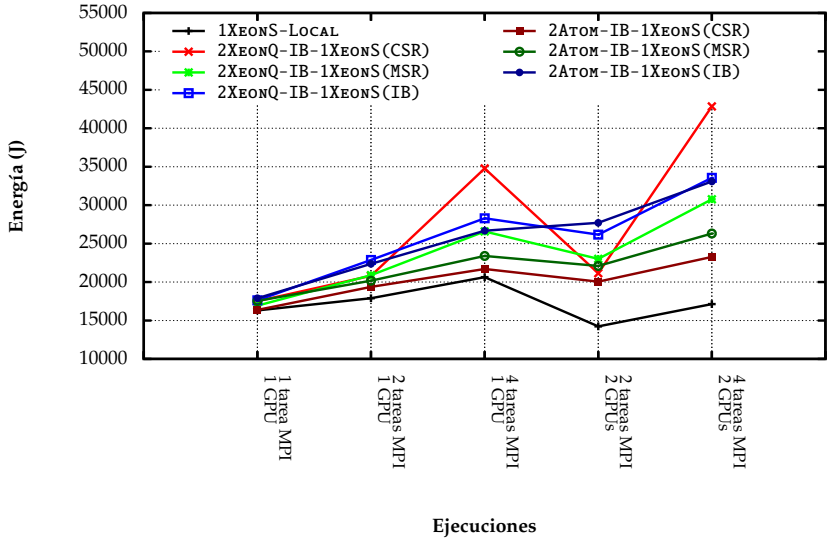
Figura 4.27: Factor de mejora (*speed-up*) de las configuraciones `1XeonS-Local` y `2Atom-IB-1XeonS` respecto a una simulación de LAMMPS utilizando 1 tarea MPI acelerada con 1 GPU sobre `1XeonS-Local`.

cómo las ejecuciones en `2Jetson-1GbE-1XeonS` que despliegan 2 tareas MPI sobre 1 nodo y acceden a 1 GPU remota consumen 8.220 J menos que en `2XeonQ-1GbE-1XeonS` siendo únicamente 0,36 s más rápidas.

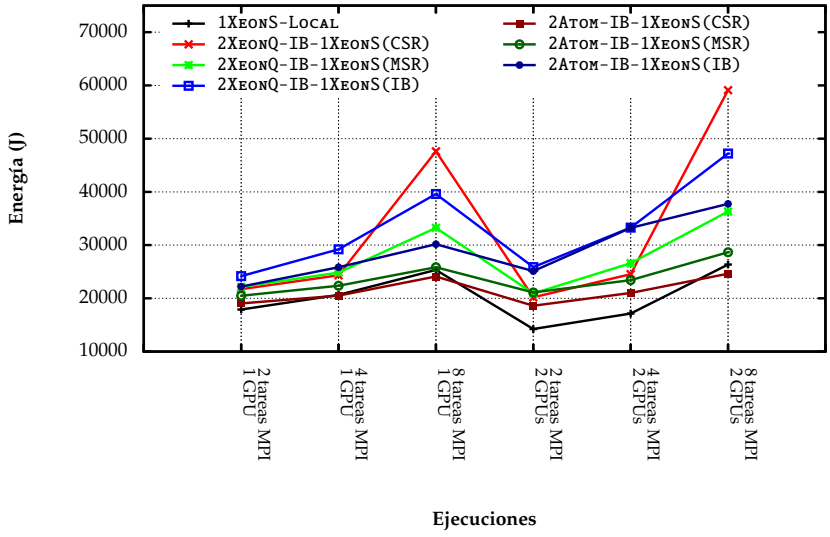
Las Figuras 4.28, 4.29 y 4.30 muestran que el módulo más eficiente para las configuraciones con redes RDMA y plataformas `ATOM` es el módulo CSR. Destaca especialmente el ahorro obtenido en las simulaciones con 8 tareas MPI sobre `2Atom-IB-1XeonS`. En particular, al emplear 1 GPU, se consigue reducir el consumo de `1XeonS-Local` en 1.223 J, lo que supone un ahorro del 4,83 %; y en 1.708 J para 2 GPUs, que representa un 6,49 %.

Por otra parte, como se aprecia en la Tabla 4.12, la potencia requerida por las configuraciones no presenta grandes diferencias cuando se utilizan los módulos CSR y MSR. En cambio, el módulo IB es capaz de obtener una ligera reducción de potencia que está parcialmente motivada por los tiempos de inactividad generados por las comunicaciones del módulo.

En el caso de la configuraciones con infraestructura de red 1GbE (Figura 4.31), la configuración `2Jetson-1GbE-1XeonS` presenta la mayor eficiencia energética de todas las configuraciones con rCUDA, llegando,



(a) Sobre un único nodo cliente



(b) Sobre dos nodos cliente

Figura 4.28: Energía (EtS) requerida por las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red IB.

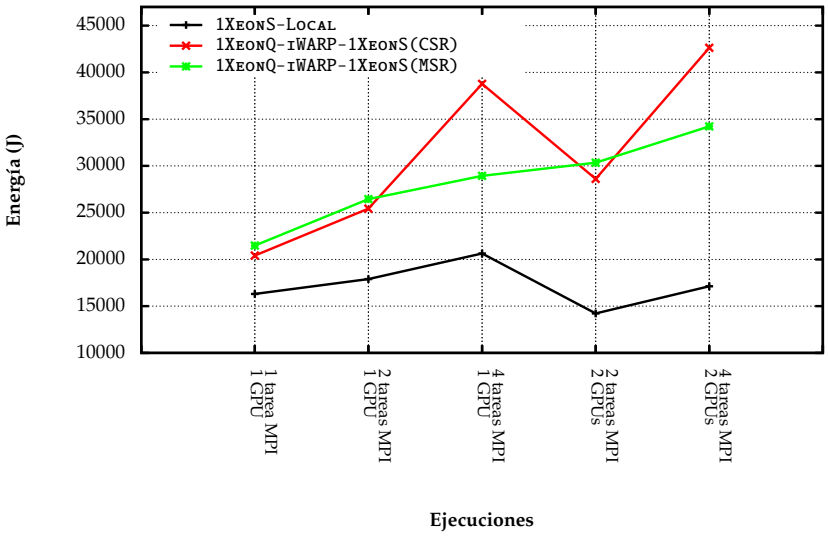


Figura 4.29: Energía (EtS) requerida por las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red iWARP.

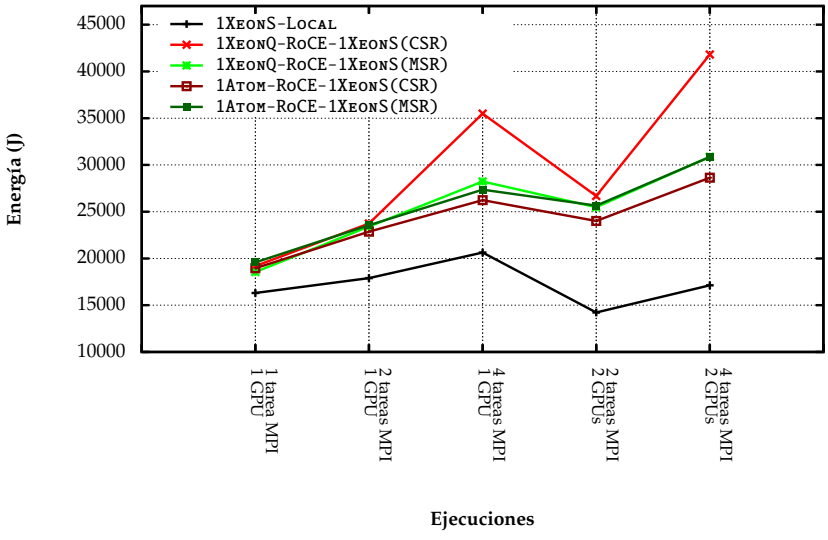
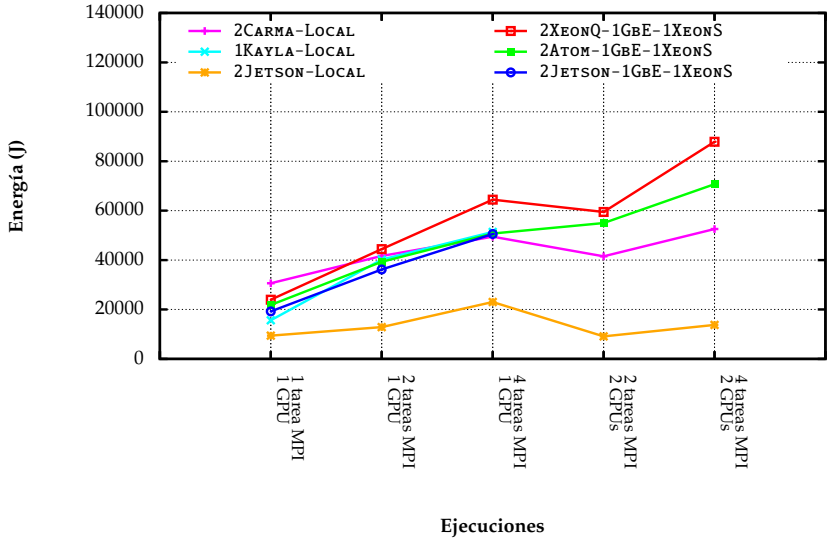
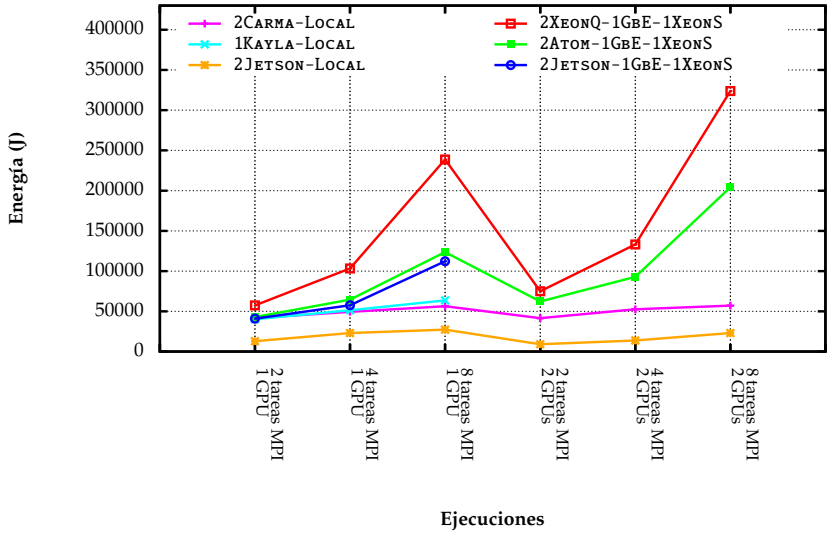


Figura 4.30: Energía (EtS) requerida por las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red RoCE.



(a) Sobre un único nodo cliente



(b) Sobre dos nodos cliente

Figura 4.31: Energía (EtS) requerida por las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas a través de una red 1GbE.

Escenario	Número de tareas MPI										
	1 nodo					2 nodos					
	1 GPU			2 GPUs		1 GPU			2 GPUs		
	1	2	4	2	4	2	4	8	2	4	8
2XeonQ-IB-1XeonS(CSR)	403	407	383	463	447	473	476	481	544	550	553
2XeonQ-IB-1XeonS(MSR)	397	404	392	465	463	475	470	481	550	543	557
2XeonQ-IB-1XeonS(IB)	398	386	375	448	443	448	448	458	503	509	526
2Atom-IB-1XeonS(CSR)	376	369	367	442	437	418	412	419	492	482	489
2Atom-IB-1XeonS(MSR)	375	366	361	437	434	417	413	411	492	487	483
2Atom-IB-1XeonS(IB)	370	350	344	410	402	394	388	390	459	449	447
1XeonQ-RoCE-1XeonS(CSR)	421	417	407	470	471	–	–	–	–	–	–
1XeonQ-RoCE-1XeonS(MSR)	415	410	412	472	481	–	–	–	–	–	–
1Atom-RoCE-1XeonS(CSR)	395	381	378	452	448	–	–	–	–	–	–
1Atom-RoCE-1XeonS(MSR)	389	375	374	444	441	–	–	–	–	–	–
1XeonQ-rWARP-1XeonS(CSR)	426	425	414	484	477	–	–	–	–	–	–
1XeonQ-rWARP-1XeonS(MSR)	423	414	412	471	487	–	–	–	–	–	–
2XeonQ-1GbE-1XeonS(TCP)	375	344	342	402	397	402	401	427	458	461	489
2Atom-1GbE-1XeonS(TCP)	351	313	308	370	369	342	338	327	402	396	389
2Jetson-1GbE-1XeonS(TCP)	312	286	280	–	–	291	285	279	–	–	–

Tabla 4.12: Potencia media (HLP) requerida por las configuraciones con virtualización de GPUs para completar la ejecución de LAMMPS utilizando varias tareas MPI cooperativas aceleradas con GPUs remotas (en W).

en ciertos casos, a presentar un consumo menor que 2Carma-Local. Por ejemplo, al ejecutar 1 tarea MPI de LAMMPS acelerada con 1 GPU, 2Jetson-1GbE-1XeonS consume el 62,77 % de la energía utilizada por 2Carma-Local. Sin embargo, cuando las tareas MPI se distribuyen entre las 2 plataforma Jetson, el consumo aumenta ya que el tiempo de ejecución se incrementa.

Por último, en la Tabla 4.12 se observa que, aumentando el número de tareas MPI, la potencia media requerida por las configuraciones con 1GbE disminuye, al contrario que la configuraciones con redes RDMA. Nuevamente, la red de interconexión hace que la CPU y la GPU permanezcan más tiempo inactivas o a la espera de que se produzcan eventos de comunicación.

4.6.3. Conclusiones

La evaluación de la aplicación LAMMPS junto con USER-CUDA han permitido explorar el rendimiento y la eficiencia energética de los escenarios con virtualización frente a la ejecución de varias tareas MPI cooperativas

que comparten la misma GPU. Este escenario puede ser considerado el peor de los casos, ya que las tareas cooperativas con un alto grado de sincronización compiten por el acceso al canal de comunicación con la GPU, lo que limita la escalabilidad. Sin embargo, gracias a que rCUDA proporciona la posibilidad de repartir el trabajo entre los nodos cliente, se produce un patrón de comunicación más desincronizado entre las tareas MPI, que reduce el problema y aumenta el rendimiento con respecto a las ejecuciones aceleradas con GPUs locales. Esta situación se observa al ejecutar de 8 tareas MPI de LAMMPS sobre la configuración `2Atom-IB-1XeonS` utilizando CSR.

Los resultados obtenidos han puesto de manifiesto el gran potencial de los módulos desarrollados en el presente trabajo en combinación con las tecnologías de bajo consumo basadas en plataformas Intel Atom. Concretamente, `2Atom-IB-1XeonS` junto a CSR han reducido el consumo de la ejecución de 8 tareas MPI de LAMMPS sobre 2 nodos acelerados con 2 GPUs un 4,83 %, añadiendo un pequeño sobrecoste en el tiempo de ejecución de un 1,67 %.

4.7. Evaluación de CUDASW++

CUDASW++ (*CUDA GPU accelerated Smith-Waterman algorithm*) es una herramienta bioinformática que recurre a la capacidad de cómputo de las GPUs para ejecutar búsquedas de secuencias en bases de datos de proteínas utilizando el algoritmo de Smith-Waterman [135, 136]. En particular, el código de CUDASW++ implementa un paralelismo que consiste en una sola hebra en la CPU asociada a una GPU [135].

CUDASW++ destaca por su alta sensibilidad en las búsquedas [137], por su rapidez [138] y por soportar patrones de búsqueda de hasta 59 KB de longitud [139].

En el presente estudio se ha utilizado la versión 2.0.10 de CUDASW++. Para evaluar los escenarios al ejecutan varias aplicaciones independientes aceleradas con la misma GPU, el código de CUDASW++ ha sido extendido con las siguientes modificaciones:

- Se ha alterado la función principal de la aplicación para que la tarea de localización de secuencias se repita durante al menos 60 segundos.

- Mediante la biblioteca de OpenMPI, se ha dotado a CUDASW++ con la capacidad de lanzar varias tareas MPI que realizan búsquedas independientes.
- Se ha insertado el código necesario para repartir equitativamente las tareas MPI entre las GPUs disponibles. Por defecto, CUDASW++ utiliza la GPU con menor el identificador CUDA del sistema.

Los experimentos han consistido en la ejecución de varias tareas MPI de CUDASW++ aceleradas con GPUs sobre las distintas configuraciones. Cada tarea MPI ha efectuado la búsqueda de la secuencia `P010008.fasta` [140] en la base de datos de proteínas `uniprot_sprot.fasta` [141] de forma independiente. Se han asignado un máximo de 2 tareas MPI por GPU. Además, en las configuraciones con dos nodos clientes, se han realizado las pruebas ubicando las tareas MPI en un mismo nodo y distribuyendo las tareas entre los dos nodos de la configuración.

4.7.1. Aceleración mediante GPUs locales

En esta evaluación, las tareas MPI de CUDASW++ se han ejecutado sobre las configuraciones de la Tabla 4.4 y han sido aceleradas mediante las GPUs locales (método tradicional de ejecución). A partir de los resultados obtenidos se ha calculado la eficiencia paralela (PEff) de cada escenario, tomando como referencia el rendimiento proporcionado por una única tarea MPI de CUDASW++ ejecutada en 1XeonS-Local (Tabla 4.13).

Escenario	Número de tareas MPI			
	1 GPU		2 GPUs	
	1	2	2	4
1XeonS-Local		0,63	0,97	0,61
2Carma-Local	0,12	0,08	0,12	0,08
2Jetson-Local	0,15	0,09	0,15	0,09
1Kayla-Local	0,18	0,15	–	–

Tabla 4.13: Eficiencia paralela (PEff) de las configuraciones sin virtualización de GPUs al ejecutar varias tareas MPI de CUDASW++ independientes aceleradas con las GPUs locales (valores normalizados con la ejecución en 1XeonS-Local de una tarea MPI de CUDASW++ acelerada con una GPU local).

Para empezar, resalta en la Tabla 4.13 el bajo rendimiento de las configuraciones 2Carma-Local, 2Jetson-Local y 1Kayla-Local, como se pone

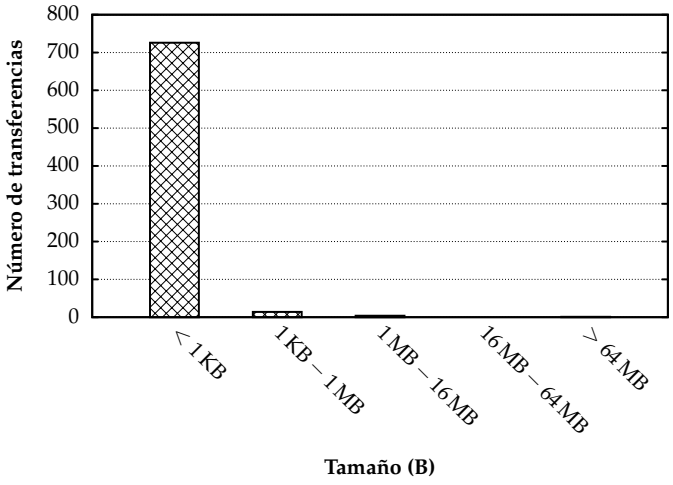


Figura 4.32: Estudio del tamaño de las transferencias efectuadas por rCUDA durante la ejecución de CUDASW++.

de manifiesto al comparar los tiempos de ejecución de 1 tarea MPI de CUDASW++ sobre estas configuraciones (34,92 s, 22,43 s y 26,83 s, respectivamente) frente a 1XeonS-Local (4,13 s). Evidentemente, este bajo rendimiento se debe al menor ancho de banda del bus PCIe de las plataformas Carma, Kayla y Jetson.

Con el objetivo de confirmar que el rendimiento del bus PCIe es la causa de este incremento en el tiempo de ejecución, se ha realizado el estudio de las transferencias efectuadas por 1 tarea MPI de CUDASW++ entre la CPU/cliente y la GPU/servidor utilizando el software de rCUDA. Los resultados de este experimento independiente, recogidos en la Figura 4.32, muestran que el 98,07 % de las trasferencias realizadas son de tamaño menor a 1 KB. En la Figura 4.1 de la Sección 4.2 se observa que las transferencias de ese tamaño entre la CPU y la GPU en XeonS son 4,39 × y 9,08 × más rápidas que en Carma/Kayla y Jetson, respectivamente.

Por otro lado, la comparación de la eficiencia ofrecida por las configuraciones 2Carma-Local y 2Jetson-Local muestra el impacto negativo de utilizar una GPU con un número reducido de núcleos.

Desde el punto de vista de la potencia (importante en un entorno de potencia limitada) 2Jetson-Local presenta un menor consumo, que en el

Escenario	Número de tareas MPI			
	1 GPU		2 GPUs	
	1	2	2	4
1XeonS-Local	642	563	514	475
2Carma-Local	1.095	921	806	706
2Jetson-Local	169	219	179	215
1Kayla-Local	1.376	906	–	–

Tabla 4.14: Energía (EtS) requerida por las configuraciones sin virtualización de GPUs para completar una tarea MPI de CUDASW++ acelerada con una GPU local (en J).

Escenario	Número de tareas MPI			
	1 GPU		2 GPUs	
	1	2	2	4
1XeonS-Local	322	338	408	446
2Carma-Local	37	41	57	65
2Jetson-Local	13	16	27	32
1Kayla-Local	135	142	–	–

Tabla 4.15: Potencia media (HLP) requerida por las configuraciones sin virtualización de GPUs para ejecutar las tareas MPI de CUDASW++ aceleradas con GPUs locales (en W).

mejor de los casos es un factor 24,77 × menor con respecto a 1XeonS-Local, 10,38 × frente a 1Kayla-Local y 2,85 × en relación con 2Carma-Local. Atendiendo a la cantidad de energía consumida para completar una búsqueda, la configuración 2Jetson-Local sigue siendo la que mejores resultados ofrece. En este caso las mejores prestaciones de 1XeonS-Local hacen que esta configuración sea claramente superior a 2Carma-Local y 1Kayla-Local, aunque la potencia media sea mayor. Por ejemplo, para la ejecución de 1 tarea MPI asignada a 1 GPU, su consumo se reduce en un factor de 0,58 × y 0,46 ×, respectivamente.

4.7.2. Aceleración mediante GPUs remotas

En este apartado se examinan los resultados obtenidos al ejecutar varias tareas MPI de CUDASW++ sobre las configuraciones con virtualización de GPUs.

En primer lugar, el análisis de los datos recogidos en las Tablas 4.16, 4.17 y 4.18 muestra que las configuraciones que utilizan tecnologías RDMA no presentan diferencias de rendimiento ni de consumo al utilizar los

Escenario	Número de tareas MPI						
	1 nodo				2 nodos		
	1 GPU		2 GPUs		1 GPU	2 GPUs	
	1	2	2	4	2	2	4
2XeonQ-IB-1XeonS(CSR)	0,93	0,59	0,89	0,57	0,59	0,91	0,58
2XeonQ-IB-1XeonS(MSR)	0,93	0,61	0,89	0,56	0,61	0,91	0,58
2XeonQ-IB-1XeonS(IB)	0,92	0,60	0,89	0,56	0,60	0,89	0,58
2Atom-IB-1XeonS(CSR)	0,71	0,49	0,68	0,45	0,50	0,69	0,48
2Atom-IB-1XeonS(MSR)	0,71	0,50	0,68	0,46	0,51	0,68	0,48
2Atom-IB-1XeonS(IB)	0,71	0,50	0,69	0,47	0,50	0,70	0,48
1XeonQ-RoCE-1XeonS(CSR)	0,91	0,58	0,85	0,55	–	–	–
1XeonQ-RoCE-2XeonS(MSR)	0,91	0,59	0,85	0,53	–	–	–
1Atom-RoCE-1XeonS(CSR)	0,70	0,48	0,66	0,44	–	–	–
1Atom-RoCE-1XeonS(MSR)	0,70	0,49	0,65	0,45	–	–	–
1XeonQ-iWARP-1XeonS(CSR)	0,91	0,59	0,86	0,56	–	–	–
1XeonQ-iWARP-1XeonS(MSR)	0,90	0,59	0,86	0,55	–	–	–
2XeonQ-1GbE-1XeonS(TCP)	0,65	0,39	0,50	0,28	0,40	0,49	0,29
2Atom-1GbE-1XeonS(TCP)	0,54	0,35	0,42	0,25	0,35	0,42	0,26
2Jetson-1GbE-1XeonS(TCP)	0,51	0,33	–	–	0,34	–	–
2XeonQ-1GbE-2Jetson(TCP)	0,18	0,10	0,16	0,09	0,10	0,18	0,09
2Atom-1GbE-2Jetson(TCP)	0,16	0,09	0,15	0,09	0,09	0,16	0,09
2XeonQ-1GbE-2Carma(TCP)	0,15	0,09	–	–	–	–	–
2XeonQ-1GbE-1Kayla(TCP)	0,28	0,17	–	–	–	–	–

Tabla 4.16: Eficiencia paralela (PEff) de las configuraciones con virtualización de GPUs al ejecutar varias tareas MPI de CUDASW++ independientes aceleradas con las GPUs remotas (valores normalizados con la ejecución en 1XeonS-Local de una tarea MPI de CUDASW++ acelerada con una GPU local).

distintos módulos de comunicación de rCUDA. La desviación relativa media es del 0,24 %.

Por otra parte, en la Tabla 4.16 es visible la superioridad de las configuraciones con plataformas XeonQ frente a Atom y Jetson. Así, por ejemplo, en el caso de las ejecuciones de 2 tareas MPI, donde cada una está asociada a un nodo cliente y asignada a una GPU remota diferente, la eficiencia respecto a la ejecución de 1 tarea MPI de CUDASW++ acelerada con 1 GPU local en 1XeonS-Local obtenida por 2XeonQ-IB-1XeonS(CSR) (que logra una eficiencia de 0,91) es 1,32 × mayor que 2Atom-IB-1XeonS(CSR) (cuyo valor es 0,69). El hecho de que esta pauta se repita en todas las tecnologías de interconexión indica que la responsabilidad del bajo rendimiento de las configuraciones recae en la misma CPU del cliente.

Otro rasgo destacable es la pérdida de eficiencia que surge al aumentar el número de tareas MPI de CUDASW++. Se observa claramente que,

Escenario	Número de tareas MPI						
	1 nodo				2 nodos		
	1 GPU		2 GPUs		1 GPU	2 GPUs	
	1	2	2	4	2	2	4
2XeonQ-IB-1XeonS(CSR)	640	583	550	518	633	580	505
2XeonQ-IB-1XeonS(MSR)	598	566	545	498	585	548	498
2XeonQ-IB-1XeonS(IB)	599	561	532	503	592	562	504
2Atom-IB-1XeonS(CSR)	701	585	592	517	594	593	511
2Atom-IB-1XeonS(MSR)	706	575	592	510	585	590	502
2Atom-IB-1XeonS(IB)	667	533	613	505	584	598	502
1XeonQ-RoCE-1XeonS(CSR)	699	628	596	548	–	–	–
1XeonQ-RoCE-2XeonS(MSR)	667	607	597	551	–	–	–
1Atom-RoCE-1XeonS(CSR)	803	672	636	573	–	–	–
1Atom-RoCE-1XeonS(MSR)	795	646	629	552	–	–	–
1XeonQ-iWARP-1XeonS(CSR)	656	601	581	525	–	–	–
1XeonQ-iWARP-1XeonS(MSR)	673	582	568	516	–	–	–
2XeonQ-1GbE-1XeonS(TCP)	868	789	846	809	873	897	840
2Atom-1GbE-1XeonS(TCP)	931	781	882	789	824	892	759
2Jetson-1GbE-1XeonS(TCP)	894	754	–	–	754	–	–
2XeonQ-1GbE-2Jetson(TCP)	260	291	250	300	341	263	316
2Atom-1GbE-2Jetson(TCP)	177	225	176	207	253	222	217
2XeonQ-1GbE-2Carma(TCP)	955	1.000	–	–	–	–	–
2XeonQ-1GbE-1Kayla(TCP)	1.134	1.040	–	–	–	–	–

Tabla 4.17: Energía (EtS) requerida por las configuraciones con virtualización de GPUs para completar una tarea MPI de CUDASW++ acelerada con una GPU remota (en J).

cuando 2 tareas MPI de CUDASW++ comparten una GPU remota, existe un decremento drástico de eficiencia, y que éste es relativamente leve en las ejecuciones donde las tareas no comparten la misma GPU. En el primer caso la penalización surge porque ambas tareas compiten por la misma GPU, lo que genera cuellos de botella. Sin embargo, los sobrecostes que sufren las tareas cuando no comparten la misma GPU son consecuencia de la tecnología de interconexión empleada en la configuración. De esta forma las configuraciones con redes más rápidas presentan una menor pérdida de eficiencia, como demuestra el hecho de que la diferencia en 2XeonQ-IB-1XeonS sea del 4,3 % frente al 23,1 % de 2XeonQ-1GbE-1XeonS.

Al comparar la eficiencia de las configuraciones que disponen de servidores ARM (Tabla 4.16) con los resultados obtenidos en las ejecuciones aceleradas con las GPUs locales de las plataformas ARM (Tabla 4.13), se aprecia que las configuraciones con virtualización mejoran el rendimiento de las configuraciones que emplean esos mismos servidores

Escenario	Número de tareas MPI						
	1 nodo				2 nodos		
	1 GPU		2 GPUs		1 GPU	2 GPUs	
	1	2	2	4	2	2	4
2XeonQ-IB-1XeonS(CSR)	382	405	475	524	477	550	577
2XeonQ-IB-1XeonS(MSR)	373	405	473	511	468	535	577
2XeonQ-IB-1XeonS(IB)	372	401	468	512	466	538	577
2Atom-IB-1XeonS(CSR)	342	361	417	449	407	460	499
2Atom-IB-1XeonS(MSR)	342	359	415	450	405	457	496
2Atom-IB-1XeonS(IB)	336	351	425	452	405	464	496
1XeonQ-RoCE-1XeonS(CSR)	399	420	489	535	–	–	–
1XeonQ-RoCE-2XeonS(MSR)	391	418	491	530	–	–	–
1Atom-RoCE-1XeonS(CSR)	357	378	425	467	–	–	–
1Atom-RoCE-1XeonS(MSR)	355	374	419	459	–	–	–
1XeonQ-rWARP-1XeonS(CSR)	385	411	483	524	–	–	–
1XeonQ-rWARP-1XeonS(MSR)	387	406	476	512	–	–	–
2XeonQ-1GbE-1XeonS(TCP)	351	364	417	437	429	474	493
2Atom-1GbE-1XeonS(TCP)	321	332	381	394	373	417	427
2Jetson-1GbE-1XeonS(TCP)	289	301	–	–	310	–	–
2XeonQ-1GbE-2Jetson(TCP)	73	75	82	87	126	133	139
2Atom-1GbE-2Jetson(TCP)	53	56	59	63	89	96	97
2XeonQ-1GbE-2Carma(TCP)	71	77	–	–	–	–	–
2XeonQ-1GbE-1Kayla(TCP)	175	184	–	–	–	–	–

Tabla 4.18: Potencia media (HLP) requerida por las configuraciones con virtualización de GPUs para ejecutar las tareas MPI de CUDASW++ aceleradas con GPUs remotas (en W).

para efectuar las "ejecuciones tradicionales" de las tareas MPI aceleradas con las GPUs locales. Esto se ilustra claramente con las configuraciones 2XeonQ-1GbE-2Jetson y 2Jetson-Local, donde la primera ofrece hasta un 20 % más de rendimiento cuando la relación entre tareas MPI y GPU utilizadas es 1 a 1.

Desde el punto de vista del consumo energético y de la potencia media requerida, las Tabla 4.17 y 4.18 muestran que la configuración 2Atom-1GbE-2Jetson proporciona las mejores prestaciones. Concretamente, las tareas MPI de CUDASW++ requieren entre 177 J y 253 J para realizar la búsqueda de la secuencia en 2Atom-1GbE-2Jetson. Esto supone un ahorro energético respecto a las configuraciones que utilizan la plataforma XeonS de entre un 57,03 % y un 80,99 %.

Ahora bien, en el caso de las configuraciones con tecnologías RDMA, las combinaciones con plataformas XeonQ actuando como cliente ofrecen un mejor consumo. Sin embargo, a medida que el número de tareas y la

cantidad de GPUs utilizadas aumenta, la diferencia con las configuraciones que emplean Atom se reduce hasta prácticamente igualarse los consumos. Esto se observa claramente en la Tabla 4.17 donde, por ejemplo, la diferencia entre los consumos de 2XeonQ-IB-1XeonS(CSR) y 2Atom-IB-1XeonS(CSR) al ejecutar 4 tareas MPI aceleradas con 2 GPUs sobre 2 nodos cliente es de apenas 6 J.

Por último, a pesar de que los resultados previos identificaban un importante cuello de botella en la interfaz 1GbE de las plataformas Carma y Kayla, se han evaluado, parcialmente y con propósito ilustrativo, los escenarios que cuentan con estas máquinas. Como se recoge en las Tablas 4.16 y 4.17, estas configuraciones con plataformas Carma y Kayla presentan una baja eficiencia debido a su considerable tiempo de ejecución, que combinado con la potencia media, conlleva el mayor coste energético.

4.7.3. Conclusiones

La aceleración mediante GPUs remotas proporciona mayores ventajas respecto a la aceleración con GPUs locales cuando el porcentaje de tiempo dedicado al cálculo paralelo es mucho mayor que el dedicado a realizar las transferencias de los datos de entrada o salida. Las pruebas efectuadas con CUDASW++ han demostrado que la aplicación realiza un reducido número de transferencias entre la CPU y la GPU, y que además son de pequeño tamaño. Como consecuencia, los sobrecostes en términos de rendimiento de la virtualización de GPUs son relativamente insignificantes. Concretamente, en la configuración 2XeonQ-IB-1XeonS empleando MSR, el tiempo de ejecución de una tarea MPI es un factor 1,07 $\times$ mayor que la ejecución tradicional, mientras que para 4 tareas MPI asignadas a 2 GPUs remotas y distribuidas en 2 nodos cliente es 1,05 $\times$ veces mayor.

Al mismo tiempo rCUDA proporciona una reducción del coste energético de la configuración a cambio de introducir este ligero sobrecoste en el tiempo de ejecución de la aplicación, que en el caso de la configuración anterior con el módulo MSR es de un nada despreciable 6,9 % al ejecutar 1 tarea MPI, siendo la mejor alternativa con virtualización de GPUs para reducir el consumo.

Para finalizar, al comparar los resultados obtenidos por las ejecuciones aceleradas con GPUs locales en las plataformas ARM y las ejecuciones

aceleradas con GPUs remotas en servidores con arquitectura ARM, se observa que las segundas mejoran el rendimiento a cambio de introducir una penalización en términos de potencia eléctrica. Así, por ejemplo, 2Atom-1GbE-2Jetson reduce un 6,3 % el tiempo de ejecución obtenido en 2Jetson-Local, pero aumenta la potencia media consumida 4,07 × veces.

4.8. Evaluación de GPU-BLAST

GPU-BLAST (*GPU Basic Local Alignment Search Tool*) es la última de las aplicaciones de producción empleada para desarrollar este estudio. Actualmente se encuentra entre las aplicaciones científicas paralelas más utilizadas en el ámbito de la bioinformática. Su impacto en este ámbito ha sido corroborado por la existencia de más de 110.000 referencias en artículos de investigación [142].

GPU-BLAST implementa una versión mejorada del algoritmo bioinformático NCBI-BLAST (*National Center for Biotechnology Information Basic Local Alignment Search Tool*), que compara una secuencia, ya sea de ADN, ARN o de proteínas, contra una gran cantidad de secuencias almacenadas en una base de datos. En particular, GPU-BLAST despliega múltiples hebras sobre la CPU que trabajan en paralelo con una GPU para realizar la búsqueda [143]. Esto permite que GPU-BLAST sea entre 3 × y 4 × veces más rápida que NCBI-BLAST.

Para llevar a cabo la evaluación de los escenarios de experimentación se ha modificado la versión "gpu-blast-1.1_ncbi-blast-2.2.28" de GPU-BLAST, de modo que varias tareas MPI independientes aceleradas con GPUs se ejecutan concurrentemente durante al menos 60 segundos.

La carga de prueba ha sido generada utilizando las baterías de pruebas incluidas en el paquete de distribución del software de GPU-BLAST. Concretamente se ha configurado la aplicación para que efectúe la búsqueda de la secuencia `SequenceLength_00000100` en la base de datos `sorted_env_nr`, que por su dimensión ha limitado a 2 el número de tareas MPI asignadas a cada GPU.

Asimismo, las pruebas preliminares con GPU-BLAST han revelado que la aplicación escoge en tiempo de ejecución cualquier GPU del sistema que disponga de suficiente memoria libre para albergar los datos. Esto no asegura que las tareas MPI se repartan equitativamente entre las GPUs

de los equipos con varios aceleradores gráficos, como es el caso de la plataforma XeonS. Por tanto, para evitar estas situaciones, las pruebas han consistido en ejecuciones de 1 y 4 tareas MPI asignadas a 1 y 2 GPUs respectivamente, que en función de los escenarios se han distribuido entre 1 o 2 nodos.

4.8.1. Aceleración mediante GPUs locales

A continuación se presenta la evaluación de la aplicación GPU-BLAST acelerada con las GPUs locales de las configuraciones incluidas en la Tabla 4.4. Los resultados obtenidos se muestran en las Tablas 4.19, 4.20 y 4.21. En concreto, la Tabla 4.19 contiene la eficiencia paralela (PEff) de cada escenario tomando como referencia el rendimiento proporcionado por 1 tarea MPI de GPU-BLAST acelerada con 1 GPU local ejecutada en 1XeonS-Local; mientras que las Tablas 4.20 y 4.21 muestran los datos referentes al consumo energético y a la potencia media requerida por las ejecuciones de GPU-BLAST.

Escenario	Número de tareas MPI	
	1	4
1XeonS-Local		0,65
2Carma-Local	0,01	0,01
2Jetson-Local	0,03	0,03
1Kayla-Local	0,02	–

Tabla 4.19: Eficiencia paralela (PEff) de las configuraciones sin virtualización de GPUs al ejecutar varias tareas MPI de GPU-BLAST independientes aceleradas con las GPUs locales (valores normalizados con la ejecución en 1XeonS-Local de una tarea MPI de GPU-BLAST acelerada con una GPU local).

Atendiendo al rendimiento computacional, en la Tabla 4.19 se observa claramente que la configuración con mejores prestaciones es 1XeonS-Local, donde la ejecución de 1 tarea MPI de GPU-BLAST acelerada mediante 1 GPU tarda 6,22 s. También se aprecia el pobre rendimiento alcanzado en las ejecuciones sobre las configuraciones con plataformas ARM. Incluso en el caso de 1 tarea MPI acelerada con 1 GPU, el rendimiento de la configuración 1Kayla-Local es 0,3 $\times$ veces el alcanzado en 1XeonS-Local, a pesar de que ambas configuraciones disponen de la misma GPU. El estudio de las transferencias efectuadas por 1 tarea MPI de GPU-BLAST, presentado en la Figura 4.33, muestra que el 93,84 % de los datos intercambiados son de

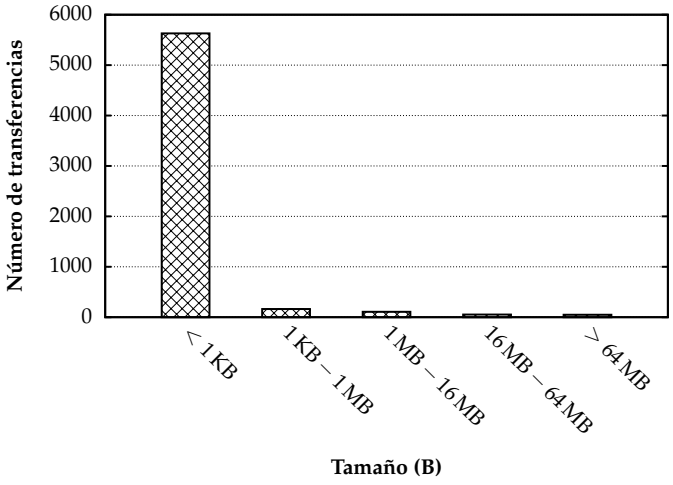


Figura 4.33: Estudio del tamaño de las transferencias efectuadas por rCUDA durante la ejecución de GPU-BLAST.

tamaño menor que 1 KB. Las pruebas ejecutadas en la Sección 4.2 indican que, en estos casos, las plataformas ARM presentan un ancho banda 9,27 × veces menor que 1XeonS-Local, mientras que los tiempos de ejecución de estas configuraciones son 33,3 × veces mayores. Estos resultados confirman que GPU-BLAST realiza un uso más intensivo de la CPU que el resto de aplicaciones de producción descritas en este capítulo.

Escenario	Número de tareas MPI	
	1	4
1XeonS-Local	270	302
2Carma-Local	30	46
2Jetson-Local	11	28
1Kayla-Local	131	–

Tabla 4.20: Potencia media (HLP) requerida por las configuraciones sin virtualización de GPUs para ejecutar las tareas MPI de GPU-BLAST aceleradas con GPUs locales (en W).

Por otra parte, desde la perspectiva de la potencia media (Tabla 4.20), 1XeonS-Local requiere 270 W para ejecutar 1 tarea MPI y 302 W para 4 tareas MPI, lo que la convierte en la configuración con mayores requisitos de potencia. A su vez, 2Jetson-Local continúa siendo la mejor opción si se establece un entorno de potencia limitada. Su potencia media va desde 11 W

Escenario	Número de tareas MPI	
	1	4
1XeonS-Local	647	325
2Carma-Local	10.465	4.322
2Jetson-Local	821	679
1Kayla-Local	22.951	–

Tabla 4.21: Energía (EtS) requerida por las configuraciones sin virtualización de GPUs para completar una tarea MPI de GPU-BLAST acelerada con una GPU local (en J).

para 1 tarea a 28 W cuando ejecuta 4 tareas, 24,54 × y 10,78 × menos que 1XeonS-Local.

Sin embargo, combinando el tiempo de ejecución y la potencia para obtener la energía (Tabla 4.21), las mejores prestaciones de la configuración 1XeonS-Local hacen que sea claramente superior a las configuraciones con plataformas Carma, Jetson y Kayla, que en el caso de 2 tareas MPI compartiendo 1 GPU es un factor de 14,85 ×, 1,52 × y 30,81 ×, respectivamente.

4.8.2. Aceleración mediante GPUs remotas

En este apartado se analizan los sobrecostes y el potencial de los diferentes escenarios con virtualización al ejecutar varias tareas MPI de GPU-BLAST aceleradas con GPUs remotas. Los resultados obtenidos en esta evaluación se muestran en las Tablas 4.22, 4.24 y 4.25.

El primer aspecto que se aprecia en la Tabla 4.22, donde se presenta la eficiencia paralela normalizada con la ejecución en 1XeonS-Local de 1 tarea MPI acelerada mediante 1 GPU local, es que el rendimiento obtenido es mayor cuando la plataforma XeonQ actúa como cliente. Basta con examinar los valores de las configuraciones con infraestructuras 1GbE para observar la gran superioridad de XeonQ (con una eficiencia de 0,28) sobre Atom y Jetson (cuya eficiencia es 0,23 y 0,03 , respectivamente). Evidentemente, la mejora del rendimiento depende en gran medida de la potencia de cálculo de la CPU equipada en la plataforma. De ahí que las configuraciones con XeonQ ofrezcan los mejores resultados, que están muy alejados de los obtenidos con los clientes Jetson. En cualquier caso, los resultados presentan una desviación relativa inferior al 8,44 %.

Por otra parte, una comparativa entre la eficiencia de los escenarios con

Escenario	Número de tareas MPI		
	1 nodo		2 nodos
	1	4	4
2XeonQ-IB-1XeonS(CSR)	0,75	0,59	0,64
2XeonQ-IB-1XeonS(MSR)	0,80	0,65	0,69
2XeonQ-IB-1XeonS(IB)	0,82	0,66	0,69
2Atom-IB-1XeonS(CSR)	0,47	0,39	0,42
2Atom-IB-1XeonS(MSR)	0,49	0,40	0,45
2Atom-IB-1XeonS(IB)	0,49	0,40	0,43
1XeonQ-RoCE-1XeonS(CSR)	0,74	0,56	–
1XeonQ-RoCE-2XeonS(MSR)	0,78	0,57	–
1Atom-RoCE-1XeonS(CSR)	0,45	0,38	–
1Atom-RoCE-1XeonS(MSR)	0,47	0,39	–
1XeonQ-rWARP-1XeonS(CSR)	0,74	0,56	–
1XeonQ-rWARP-1XeonS(MSR)	0,78	0,57	–
2XeonQ-1GbE-1XeonS(TCP)	0,28	0,11	0,11
2Atom-1GbE-1XeonS(TCP)	0,23	0,11	0,11
2Jetson-1GbE-1XeonS(TCP)	0,03	–	–
2XeonQ-1GbE-2Jetson(TCP)	0,13	0,07	0,06
2Atom-1GbE-2Jetson(TCP)	0,10	0,07	0,06
2XeonQ-1GbE-2Carma(TCP)	0,08	–	–
2XeonQ-1GbE-1Kayla(TCP)	0,08	–	–

Tabla 4.22: Eficiencia paralela (PEff) de las configuraciones con virtualización de GPUs al ejecutar varias tareas MPI de GPU-BLAST independientes aceleradas con las GPUs remotas (valores normalizados con la ejecución en 1XeonS-Local de una tarea MPI de GPU-BLAST acelerada con una GPU local).

tecnologías RDMA y clientes XeonQ muestra que las configuraciones con virtualización sobre la red IB maximizan su rendimiento utilizando el módulo de comunicaciones IB (que al ejecutar 1 tarea MPI es 0,82 comparado con 0,80 para MSR y 0,75 para CSR). En cuanto a las configuraciones con infraestructuras 10GbE, las mejores prestaciones se alcanzan con el módulo MSR (0,78 frente a 0,74 de CSR).

Asimismo, en la Tabla 4.22 destaca negativamente el rendimiento de las configuraciones con tecnologías RDMA cuando se utiliza el módulo CSR. Esto podría ser debido a la existencia de transferencias de gran tamaño durante la ejecución de la aplicación GPU-BLAST, tal como se advierte en la Figura 4.33. En particular, el análisis detallado de la traza proporcionada por el software de rCUDA revela la existencia de 58 transferencias de datos con tamaño mayor a los 50 MB y 42 con tamaños superiores a 100 MB.

Para confirmar que el tamaño de los datos transferidos entre el cliente y el servidor de rCUDA es la causa de la reducción del rendimiento

Escenario	Tamaño de la transferencia (MB)	
	64	128
2XeonQ-IB-1XeonS(CSR)	2.392,05	2.095,65
2XeonQ-IB-1XeonS(MSR)	2.132,60	2.140,01
2XeonQ-IB-1XeonS(IB)	2.742,30	2.718,10

Tabla 4.23: Comparativa del ancho de banda proporcionado por `bandwidthTest` para los módulos de comunicación de rCUDA al efectuar transferencias de 64 MB y 128 MB en 2XeonQ-IB-1XeonS.

del módulo CSR, se ha realizado una prueba utilizando la aplicación `bandwidthTest`. Concretamente, se ha evaluado el rendimiento de la configuración 2XeonQ-IB-1XeonS al ejecutar transferencias de 64 MB y 128 MB entre la CPU y la GPU del servidor. Los resultados de este experimento, recogidos en la Tabla 4.23, indican que el ancho de banda obtenido con módulo CSR al efectuar las transferencias de 128 MB se reduce aproximadamente un 13 % respecto a las transferencias de 64 MB. En cambio, con los otros dos módulos de comunicación se mantiene estable.

Escenario	Número de tareas MPI		
	1 nodo		2 nodos
	1	4	4
2XeonQ-IB-1XeonS(CSR)	766	390	395
2XeonQ-IB-1XeonS(MSR)	724	355	355
2XeonQ-IB-1XeonS(IB)	685	344	347
2Atom-IB-1XeonS(CSR)	927	391	361
2Atom-IB-1XeonS(MSR)	910	367	336
2Atom-IB-1XeonS(IB)	960	358	336
1XeonQ-RoCE-1XeonS(CSR)	796	416	–
1XeonQ-RoCE-2XeonS(MSR)	733	387	–
1Atom-RoCE-1XeonS(CSR)	1.087	419	–
1Atom-RoCE-1XeonS(MSR)	1.085	413	–
1XeonQ-rWARP-1XeonS(CSR)	888	429	–
1XeonQ-rWARP-1XeonS(MSR)	899	302	–
2XeonQ-1GbE-1XeonS(TCP)	2.248	1.183	1.696
2Atom-1GbE-1XeonS(TCP)	2.177	1.247	1.309
2Jetson-1GbE-1XeonS(TCP)	13.179	–	–
2XeonQ-1GbE-2Jetson(TCP)	406	463	668
2Atom-1GbE-2Jetson(TCP)	135	179	183
2XeonQ-1GbE-2Carma(TCP)	2.372	–	–
2XeonQ-1GbE-1Kayla(TCP)	6.131	–	–

Tabla 4.24: Energía (EtS) requerida por las configuraciones con virtualización de GPUs para completar una tarea MPI de GPU-BLAST acelerada con una GPU remota (en J).

Escenario	Número de tareas MPI		
	1 nodo		2 nodos
	1	4	4
2XeonQ-IB-1XeonS(CSR)	326	391	453
2XeonQ-IB-1XeonS(MSR)	333	386	451
2XeonQ-IB-1XeonS(IB)	329	383	450
2Atom-IB-1XeonS(CSR)	292	320	360
2Atom-IB-1XeonS(MSR)	293	316	359
2Atom-IB-1XeonS(IB)	297	314	354
1XeonQ-RoCE-1XeonS(CSR)	338	395	–
1XeonQ-RoCE-2XeonS(MSR)	336	385	–
1Atom-RoCE-1XeonS(CSR)	300	326	–
1Atom-RoCE-1XeonS(MSR)	302	322	–
1XeonQ-rWARP-1XeonS(CSR)	346	397	–
1XeonQ-rWARP-1XeonS(MSR)	352	351	–
2XeonQ-1GbE-1XeonS(TCP)	314	353	381
2Atom-1GbE-1XeonS(TCP)	279	290	324
2Jetson-1GbE-1XeonS(TCP)	252	–	–
2XeonQ-1GbE-2Jetson(TCP)	71	83	139
2Atom-1GbE-2Jetson(TCP)	48	54	86
2XeonQ-1GbE-2Carma(TCP)	65	–	–
2XeonQ-1GbE-1Kayla(TCP)	168	–	–

Tabla 4.25: Potencia media (HLP) requerida por las configuraciones con virtualización de GPUs para ejecutar las tareas MPI de GPU-BLAST aceleradas con GPUs remotas (en W).

Desde la perspectiva de la potencia media requerida, cuando las tareas MPI se ejecutan exclusivamente sobre un único nodo que accede a la GPU remota a través de tecnologías RDMA, las configuraciones con plataformas XeonQ como clientes no muestran una gran desventaja frente a las configuraciones con máquinas Atom (aproximadamente entre 30 W y 75 W, o entre un 12 % y un 22 %), que combinado con la diferencia temporal, explica por qué las configuraciones con plataformas XeonQ presentan una solución energéticamente más eficiente que las configuraciones con clientes Atom. No obstante, al distribuir las tareas entre 2 nodos, la potencia media de las configuraciones con XeonQ aumenta un 17 % comparado con el 14 % de Atom. Esto ofrece una ventaja (de entre un 5 % y un 10 %) en términos de consumo energético a las configuraciones con 2 clientes Atom frente a las que disponen de 2 clientes XeonQ.

Respecto a las configuraciones con infraestructuras 1GbE, como se distingue en las Tablas 4.25 y 4.24, la configuración 2Atom-1GbE-2Jetson ofrece las mejores prestaciones energéticas. Al comparar sus resultados con

los obtenidos por la configuración 2XeonQ-1GbE-1XeonS, que presenta el mayor rendimiento, se observa que la ejecución de 1 tarea MPI acelerada con 1 GPU sobre un nodo requiere una potencia media 6,5 × veces menor y reduce el consumo un 94 % a pesar de multiplicar su tiempo de ejecución por un factor de 2,8 ×.

Por otro lado, los escenarios con plataformas Carma y Kayla muestran nuevamente un elevado consumo energético debido al considerable tiempo de ejecución que genera el cuello de botella en la interfaz red identificado en las secciones previas.

Ahora bien, los resultados recogidos en las Tablas 4.19, 4.21, 4.22 y 4.24 demuestran que las ejecuciones de GPU-BLAST sobre configuraciones con clientes Atom o XeonQ y servidores ARM presentan un tiempo de ejecución (hasta 4,3 × veces) y un consumo energético (hasta 6,1 × veces) menor que las ejecuciones tradicionales sobre las propias plataformas ARM.

Para finalizar este análisis, la comparativa entre el rendimiento de las configuraciones 2XeonQ-IB-1XeonS y 1XeonS-Local para GPU-BLAST (las Tablas 4.19 y 4.22) muestra que las ejecuciones de varias tareas MPI de GPU-BLAST con rCUDA sobre un nodo no experimentan sobrecarga respecto al uso de la GPU local, y que las ejecuciones aceleradas con GPUs remotas sobre dos nodos son inapreciablemente más rápidas. Aparentemente, la gestión de las GPUs locales efectuada por la aplicación unida al solapamiento de las transferencias de más de 64 MB realizado por las 4 tareas MPI, tiene un impacto negativo en el rendimiento del bus PCIe que comunica la CPU con las GPUs en la configuración 1XeonS-Local.

4.8.3. Conclusiones

La aplicación GPU-BLAST ha permitido evaluar el rendimiento y la eficiencia de las configuraciones hardware ante la ejecución de diferentes tareas MPI independientes que se disputan el acceso simultáneo a la misma GPU, lo que implica problemas de escalabilidad. A esto se añade que varias de las transferencias efectuadas por la aplicación son de un tamaño muy considerable, lo que produce un impacto negativo en el rendimiento del sistema de interconexión entre la CPU y la GPU. Sin embargo, en este caso, la utilización de rCUDA ha permitido generar una pauta de comunicación más secuenciada para estas transferencias de gran tamaño. Como resultado,

el rendimiento obtenido por 4 tareas MPI no sincronizadas de GPU-BLAST sobre `2XeonQ-IB-1XeonS`, utilizando el módulo de semántica de memoria desarrollado en este trabajo, ha sido 1,06 × veces mayor que el obtenido por las ejecuciones aceleradas con GPUs locales. Esto demuestra la viabilidad de utilizar la aceleración de GPUs remotas a través de redes RDMA en los problemas adecuados, como por ejemplo GPU-BLAST.

Por último, los resultados indican de nuevo que las configuraciones con virtualización utilizando servidores con arquitectura ARM mejoran notablemente el rendimiento obtenido en esas mismas plataformas ARM mediante la aceleración con las GPUs locales, a cambio de un pequeño incremento de la potencia media consumida.

4.9. Resumen y conclusiones

Este capítulo se ha centrado en la evaluación experimental de las posibilidades que el estado del arte tecnológico en el campo del HPC y las alternativas de bajo consumo ofrecen al aplicar la aceleración de aplicaciones científicas utilizando procesadores gráficos remotos.

El capítulo ha comenzado con la presentación de las diversas configuraciones CPU-GPU (cliente-servidor) elegidas para realizar la evaluación de rCUDA, y en particular de los módulos de comunicación desarrollados en el presente trabajo. Para ello se han combinado plataformas hardware que disponen de una o varias GPUs con sistemas que incorporan procesadores de propósito general Intel Xeon de 64 bits y equipos con arquitectura Intel Atom o ARM de menor potencia de cómputo y consumo energético. Las plataformas se han interconectado empleando infraestructuras de red 1GbE, iWARP, RoCE y IB, que son tecnologías representativas de las familias de interconexión más extendidas en los HPCC modernos y que están presentes en el 82,4 % de la totalidad de los sistemas de la lista TOP500 [18].

A continuación se ha realizado la caracterización de las transferencias entre la CPU/cliente y la GPU/servidor en las configuraciones anteriormente introducidas. Concretamente se ha utilizado la herramienta `bandwidthTest`, proporcionada por NVIDIA en el entorno de desarrollo de CUDA, para evaluar el impacto del bus PCIe que comunica la GPU con la CPU y del sistema de interconexión empleado por clientes y servidores rCUDA en el rendimiento general de las transferencias entre la CPU y la

GPU remota. Los resultados han revelado un importante cuello de botella en el acceso a la red por parte de las plataformas CARMA y KAYLA. Además, han evidenciado la inestabilidad de la tasa de transferencias de la plataforma JETSON a través de la red 1GbE y han puesto de manifiesto el bajo rendimiento de su PCIe para las transferencias pequeñas. Por otro lado, el estudio del rendimiento obtenido por las configuraciones con tecnologías RDMA utilizando los diferentes módulos de comunicación de rCUDA ha demostrado que el módulo CSR, introducido en este trabajo, es la mejor opción para las redes 10GbE y, cuando las transferencias son menores a 1 MB, también en las infraestructuras de red IB. Asimismo, se ha confirmado que las configuraciones formadas por plataformas con procesadores de propósito general Intel Xeon y procesadores de bajo consumo Intel Atom presentan un rendimiento muy similar cuando acceden a una GPU remota a través de infraestructuras 1GbE o 10GbE. Por contra, en las redes IB existe una leve diferencia a favor de XEONQ.

En la última parte de este capítulo se ha explorado el rendimiento y la eficiencia energética de las configuraciones mediante la ejecución de cuatro aplicaciones científicas paralelas que emplean aceleradores gráficos: HPL-Fermi, LAMMPS, CUDASW++ y GPU-BLAST. Se ha examinado el impacto de las plataformas y de las GPUs que incorporan las configuraciones utilizando un conjunto de métricas destinadas a evaluar el rendimiento y a medir el consumo energético de la ejecución de cada una las aplicaciones. Además, se ha analizado la respuesta de las configuraciones a medida que se ha aumentado el número de tareas MPI de las aplicaciones asignadas a las GPUs.

Los resultados de la evaluación del benchmark HPL-Fermi han mostrado la posibilidad de eliminar las limitaciones impuestas en la asignación de GPUs, existentes en las ejecuciones aceleradas con GPUs locales, mediante la agregación de varias GPU remotas, lo que proporciona un mayor rendimiento. Las aplicaciones LAMMPS y GPU-BLAST han permitido verificar que la virtualización remota utilizando redes de alto rendimiento es la mejor solución en condiciones de uso intensivo de la CPU y la GPU. La evaluación de la aplicación CUDASW++ ha expuesto que, al utilizar la aceleración remota, es posible reducir el consumo energético a cambio de añadir una pequeña sobrecarga en el tiempo de ejecución.

Desde un punto de vista del rendimiento, el estudio presentado

demuestra que el módulo CSR desarrollado en este trabajo es la mejor opción para las transferencias de rCUDA sobre las tecnologías de red con soporte para RDMA. Asimismo, los resultados de las aplicaciones de producción CUDASW++ y GPU-BLAST han señalado que la combinación de aceleradores remotos y plataformas cliente ATOM o XEONQ mejoran el rendimiento obtenido por la aceleración local en las plataformas JETSON, CARMA y KAYLA.

Por otro lado, desde la perspectiva del consumo de energía, los resultados de la experimentación efectuada a lo largo de este capítulo confirman que el mayor ahorro energético, a cambio de un pequeño sobrecoste temporal, se lograr en las ejecuciones sobre plataformas basadas en Intel Atom aceleradas mediante GPUs remotas.

En conclusión, la presente investigación ha demostrado que los módulos de comunicación CSR y MSR desarrollados en el presente trabajo permiten aumentar el rendimiento y la eficiencia energética de las aplicaciones paralelas aceleradas con GPU remotas respecto a los módulos existentes para rCUDA, facilitando la reducción del consumo de energía frente a configuraciones donde todos los nodos tienen al menos una GPU, que permanece ociosa una parte significativa del tiempo.

CAPÍTULO 5

Conclusiones y lineas abiertas de investigación

Este capítulo pone el punto final al documento, resumiendo y analizando el trabajo presentado. En primer lugar se presenta un breve sumario de esta tesis doctoral destacando las principales contribuciones y aportaciones. El capítulo finaliza exponiendo algunas propuestas para futuros trabajos e investigaciones.

5.1. Conclusiones y aportaciones

Actualmente, una de las tendencias más extendidas en HPC consiste en utilizar procesadores gráficos para acelerar las partes de un programa que requieren cálculo intensivo, mientras que el resto de código se ejecuta en las CPUs. Los beneficios, en términos de menor tiempo de ejecución, que aporta el uso de estos coprocesadores gráficos hace que, cada vez más, los HPCC incorporen este tipo de aceleradores en sus configuraciones. Por ejemplo, dos de los diez primeros puestos de la última lista TOP500 (a fecha de junio de 2015) incorporan aceleradores de NVIDIA. Estos dispositivos están conectados al bus PCIe de los nodos de cálculo, y únicamente puede insertarse un reducido número de estos dispositivos por nodo, ya que depende del espacio, del bus y de las limitaciones de potencia eléctrica. Generalmente en los clústeres HPC se utilizan configuraciones con una o dos GPUs, aunque algunas soluciones específicas permiten aumentar este número hasta ocho aceleradores. Las APIs actuales con soporte para GPGPU permiten una programación relativamente fácil de éstos dispositivos, siendo CUDA, la API proporcionada por NVIDIA, la más utilizada.

A pesar de los beneficios mencionados, equipar todos los nodos de un clúster con aceleradores es costoso, puesto que esto puede aumentar el consumo de energía de un HPCC cerca de un 30 % y el coste de adquisición aproximadamente un 50 %. Además, estos dispositivos permanecen ociosos una parte significativa del tiempo, ya que no todos los códigos permiten la aceleración con CPUs.

La virtualización de GPUs surge con el propósito de resolver estas cuestiones. En particular, la solución de virtualización de GPUs propuesta por rCUDA ofrece acceso a todos los aceleradores instalados en un clúster desde cualquiera de sus nodos. Por tanto, rCUDA permite construir un clúster con un número reducido de GPUs, facilitando un coste menor, tanto energético como de adquisición, con respecto a la configuración donde todos los nodos tienen al menos una GPU, que permanece ociosa un importante porcentaje del tiempo.

Otra de las estrategias para aumentar la eficiencia energética de los HPCCs y reducir costes es el uso de plataformas con procesadores de bajo consumo energético, como es el caso de los procesadores Intel Atom y ARM Cortex. Estos procesadores han sido diseñados para funcionar en entornos de potencia eléctrica y temperatura limitadas y, al mismo tiempo, incorporan unidades funcionales de cálculo que les permiten realizar operaciones en coma flotante de doble precisión, lo que les hace especialmente interesantes para HPC.

Esta tesis doctoral aborda dos carencias de estos entornos:

- El software de rCUDA únicamente soporta la virtualización para redes TCP/IP e infraestructuras IB. Al emplear cualquier tecnología de interconexión HPC diferente a IB, las transferencias entre clientes y servidores de rCUDA se realizan utilizando las comunicaciones TCP/IP. Esto limita en gran medida las prestaciones de la virtualización de GPUs, tanto en rendimiento como en términos de consumo de energía.

- Las plataformas de bajo consumo se caracterizan por presentar un consumo de potencia eléctrica reducido. Sin embargo, una potencia baja normalmente supone un rendimiento menor. En consecuencia, se incrementa el tiempo de ejecución de las aplicaciones HPC, lo que a la larga puede redundar en un mayor consumo de energía.

Teniendo en cuenta estas circunstancias, esta tesis propone y analiza un sistema de comunicaciones para la virtualización de GPUs sobre sistemas heterogéneos. Esta propuesta adapta la solución rCUDA para la virtualización de GPUs a las tecnologías de interconexión RDMA, de forma que permite a las configuraciones hardware heterogéneas, formadas por plataformas de bajo consumo interconectadas con aceleradores remotos a través de redes HPC, lograr la sinergia necesaria para abordar el reto de la reducción del consumo energético sin afectar significativamente al rendimiento.

Como aportación principal de la tesis se han diseñado y desarrollado dos soluciones software productivas que completan rCUDA con la funcionalidad necesaria para permitir a las aplicaciones de HPC acceder a aceleradores remotos a través de redes RDMA. Estos desarrollos emplean operaciones de comunicación específicas de las tecnologías RDMA, de modo que amplían y mejoran las prestaciones de rCUDA, a la vez que aumentan la eficiencia energética de las configuraciones hardware con virtualización de GPUs.

Entrando en mayor detalle, en primer lugar, se ha explorado la viabilidad de este enfoque mediante un prototipo en el que se han utilizado operaciones específicas de las tecnologías de red con soporte para el protocolo RDMA. Para el desarrollo del prototipo se ha presentado una arquitectura acorde al modelo cliente-servidor que suministra una abstracción lógica de las entidades en función de su papel en el proceso de comunicación de rCUDA. Además, con el propósito de maximizar el ancho de banda de las transferencias entre clientes y servidores de rCUDA, como parte de este trabajo se han introducido varios mecanismos diseñados específicamente para las transferencias de rCUDA sobre redes RDMA. En este sentido, los desarrollos más destacables son el doble canal de comunicación y la agrupación de transferencias en ráfagas. Una comparativa del rendimiento de este prototipo sobre una red IB frente a los módulos de rCUDA existentes ha demostrado que, a pesar de contar con ciertas carencias, el prototipo supera al módulo basado en operaciones de comunicación de TCP/IP. En cambio, las pruebas sobre las redes RoCE y iWARP han revelado la necesidad de mejorar el protocolo de comunicación entre clientes y servidores rCUDA.

La primera de las soluciones productivas desarrollada en el presente trabajo para las redes RDMA se ha diseñado utilizando las operaciones

de comunicación de la semántica de canal. Este primer módulo, denominado CSR, emplea un protocolo propio con el fin de establecer un doble canal dedicado para realizar las transferencias entre clientes y servidores de rCUDA, de modo que los datos circulan por un canal de comunicación y los mensajes de control de flujo por el otro. Además, a fin de maximizar el rendimiento de las transferencias, se ha mejorado la agrupación de transferencias, se han empleado búferes intermedios, y se han confeccionado mecanismos de notificación de recepciones con antelación y de comprobación tardía de errores.

La segunda solución completa aportada en el marco de esta tesis, el módulo MSR, se basa en las operaciones RDMA de semántica de memoria y dispone de un proceso de establecimiento del canal en el que se intercambian las direcciones de los búferes destinados a las escrituras remotas. Concretamente, se ha desarrollado un mecanismo de transmisión para el módulo MSR que integra las escrituras remotas junto a los mensajes de control y de notificación para realizar las transferencias.

El tamaño de los búferes intermedios de las transferencias de ambos módulos ha sido adaptado a cada una de las tecnologías RDMA existentes. Asimismo, se han efectuado estudios del impacto de los métodos de consulta de finalización de las operaciones RDMA sobre el rendimiento y el consumo de las transferencias realizadas empleando los módulos CSR y MSR. Los resultados han mostrado que, al usar el módulo CSR, es posible reducir el consumo un 6,3 % a cambio de introducir una pérdida de rendimiento del 4,87 %, mientras que para MSR el ahorro es del 3,9 % obteniendo un 52 % menos de rendimiento.

Un estudio preliminar del impacto de las estrategias de consulta sobre el tiempo de ejecución y el consumo de una aplicación paralela ha sido presentada en la publicación:

- M. Castillo, J. C. Fernández, R. Mayo, E. S. Quintana-Ortí y V. Roca. «Analysis of Strategies to Save Energy for Message-Passing Dense Linear Algebra Kernels». En: *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2012, págs. 346-352.

Con el objetivo de evaluar el potencial y los sobrecostes de la solución de virtualización de GPUs de rCUDA, junto con los módulos diseñados

en este trabajo, se han establecido diversas configuraciones CPU-GPU (cliente-servidor) combinando plataformas hardware que disponen de una o varias GPUs con sistemas que incorporan procesadores de propósito general Intel Xeon de 64 bits y equipos con arquitectura Intel Atom o ARM, de menor potencia de cómputo pero con mayor eficiencia energética. Las plataformas se han interconectado empleado las tecnologías más representativas de los sistemas de interconexión de los HPCC actuales: 1GbE, iWARP, RoCE y IB.

La caracterización de las transferencias entre la CPU del cliente y la GPU del servidor en las configuraciones introducidas ha revelado un importante cuello de botella en el acceso a la red para las plataformas basadas en la solución NVIDIA Tegra 3. Mientras que las plataformas basadas en el sistema NVIDIA Tegra K1 han presentado inestabilidad en las transferencias realizadas a través de la red 1GbE y un pobre rendimiento del bus PCIe que conecta la CPU y la GPU.

El estudio del ancho de banda de las transferencias para las tecnologías RDMA ha mostrado el excelente rendimiento de los módulos creados en el presente trabajo de tesis sobre este tipo de redes. Concretamente, para transferencias con tamaños de datos pequeños, el módulo CSR ha proporcionado más del doble de ancho de banda que los módulos existentes, lo que demuestra la viabilidad del enfoque presentado. Además, se ha confirmado que las configuraciones formadas por plataformas con procesadores de propósito general Intel Xeon y procesadores de bajo consumo Intel Atom presentan un rendimiento muy similar cuando acceden a una GPU remota a través de infraestructuras 1GbE o 10GbE. En cambio, en las redes IB existe una leve diferencia a favor de las plataformas con procesadores Intel Xeon. Asimismo, se ha constatado que el ancho de banda de las transferencias entre la CPU y la GPU remota a través de redes RDMA, empleando los módulos desarrollados por el autor de esta tesis, es mayor que el obtenido entre la CPU y la GPU a través del bus PCIe de las plataformas híbridas ARM.

Con el propósito de explorar las posibilidades de los futuros HPCCs caracterizados por un paralelismo heterogéneo dirigido a aumentar la eficiencia energética, se ha efectuado un análisis experimental del rendimiento y del consumo eléctrico de las ejecuciones de HPL-Fermi, LAMMPS, CUDASW++ y GPU-BLAST aceleradas con GPUs remotas frente a

la aceleración tradicional (a través de GPUs locales). El estudio ha mostrado el potencial de utilizar los módulos de comunicación CSR y MSR para efectuar la virtualización de GPUs a través de redes RDMA. Al mismo tiempo se ha hecho evidente que las ejecuciones sobre configuraciones con plataformas Intel Atom aceleradas con GPUs remotas logran el mayor ahorro energético, a cambio de un leve sobrecoste temporal. Por otro lado, los resultados han revelado que la combinación de aceleradores remotos y plataformas cliente con procesadores Intel Atom o Intel Xeon mejora el rendimiento obtenido por la aceleración local en las plataformas con arquitectura ARM.

Parte de los resultados presentados en este estudio han quedado reflejados en las siguientes publicaciones:

- Adrián Castelló, José Duato, Rafael Mayo, Antonio J. Peña, Enrique S. Quintana-Ortí, Vicente Roca y Federico Silla. «Acelerando Aplicaciones Científicas Con GPUs Remotas y Procesadores De Bajo Consumo». En: *XXV Jornadas de paralelismo*. 2014, págs. 187-193.

- Adrián Castelló, José Duato, Rafael Mayo, Antonio J. Peña, Enrique S. Quintana-Ortí, Vicente Roca y Federico Silla. «On the Use of Remote GPUs and Low-Power Processors for the Acceleration of Scientific Applications». En: *ENERGY 2014, The Fourth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*. 2014, págs. 57-62.

Las tendencias presentes en las listas TOP500 y Green500 muestran una gran variedad en los sistemas HPC, con nodos de distintos tipos específicamente elegidos para cierta clase de aplicaciones. En el futuro se debe esperar que los aceleradores hardware, bien las GPUs o arquitecturas de propósito más general como los Intel Xeon Phi, jueguen un papel importante en los sistemas Exaescala. En cualquier caso, independientemente del hardware utilizado para superar la barrera del Exaflop, resulta sensato pensar que no todos los nodos/procesos/hilos en estos sistemas HPC tendrán acceso directo a los aceleradores, por lo que será necesario algún tipo de acceso remoto. Los datos sobre el rendimiento y el consumo presentados en esta tesis son muy prometedores, y sugieren que la solución de virtualización de rCUDA, junto con los desarrollos CSR y MSR, puede

ser parte de los cimientos de esa nueva generación de clústeres de alto rendimiento basados en la computación remota. Cabe esperar que estos sistemas futuros estén caracterizados por unos consumos energéticos y de adquisición menores que las actuales configuraciones HPC donde todos los nodos tienen al menos una GPU, que permanece ociosa una parte nada despreciable de tiempo.

5.2. Lineas abiertas de investigación

La tesis cubre con sus objetivos el diseño y evaluación de un sistema de comunicaciones sobre tecnologías RDMA para la virtualización remota de aceleradores gráficos en sistemas heterogéneos. A su conclusión existen una serie de cuestiones abiertas que pueden ser objeto de un futuro trabajo o investigación:

- Como han revelado los resultados de la evaluación experimental, la solución de comunicación CSR desarrollada en este trabajo presenta mejor rendimiento que el resto de opciones evaluadas siempre que las transferencias no sean de un tamaño considerable. En cambio, cuando se supera 1 MB es preferible el módulo IB que incluye el software de virtualización de rCUDA. Por otro lado, el módulo MSR, diseñado también para la presente tesis, cuenta con las mejores prestaciones al ejecutar aplicaciones que consumen una gran cantidad de recursos de la CPU. Con la intención de sacar el máximo partido a cada una de las ventajas ofrecidas, se propone una modificación en el mecanismo de carga de las bibliotecas dinámicas de comunicación implementado en el software de rCUDA. De este modo, el software será capaz de escoger el módulo de comunicaciones dinámicamente en función de parámetros tales como la carga del sistema, el tamaño de las transferencias o el patrón de comunicación de la aplicación, entre otros.

- Al compartir GPUs entre los nodos del clúster se plantea un requisito adicional a los originales de CUDA. En concreto, se requiere un planificador de tareas global que tenga en cuenta el modo de funcionamiento de rCUDA, ya que los esquemas existentes actualmente no son adecuados para gestionar grupos de recursos independientes

de los nodos de computación. Es necesario establecer un enfoque diferente en la asignación de recursos para poder integrar rCUDA en los planificadores de trabajos de un clúster. Debido a que la asignación de un recurso de rCUDA se realiza de forma independiente a su ubicación física, es imprescindible la utilización de contadores de recursos a nivel global. Además, se proponen dos modos para la asignación una GPU remota: compartido y exclusivo. De esta manera, el modo compartido optimizará la utilización de recursos a cambio de obtener un rendimiento menor. En este caso, la memoria disponible en cada GPU puede suponer una restricción en la asignación. En cambio, el modo exclusivo priorizará el rendimiento y la limitación puede aparecer al solicitar más GPUs que las que se encuentran ociosas y disponibles en el clúster.

- En línea con lo anterior, se sugiere completar el planificador de tareas con los mecanismos necesarios para migrar dinámicamente el trabajo entre las GPUs remotas asignadas utilizando el modo compartido. Esto permitiría agrupar los trabajos sobre los aceleradores con mayor cantidad de memoria disponible y reduciría la fragmentación de los recursos. Al mismo tiempo, en función de la demanda existente, se podrían liberar GPUs que quedarían disponibles para la ejecución de los trabajos en espera. Posteriormente, cuando el número de solicitudes decreciese, podría tener lugar una nueva redistribución para evitar que existiesen recursos ociosos.

- Por otra parte, con el objetivo de cubrir las necesidades de la comunidad de usuarios HPC que disponen de plataformas con sistemas operativos Microsoft Windows, se propone el desarrollo y evaluación de las soluciones de virtualización de GPUs para redes RDMA sobre estos operativos.

- Finalmente, los resultados presentados en la evaluación de las configuraciones hardware revelan cuellos de botella en las plataformas con arquitectura ARM, que conviene indagar más profundamente mientras se espera que las nuevas tecnologías resuelvan este punto débil.

Bibliografía

[1] United States Environmental Portection Agency. *U.S. Environmental Protection Agency*. `http://www.epa.gov`, 2014.

[2] Keren Bergman y col. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. 2008.

[3] Steve Ashby y col. «The opportunities and challenges of exascale computing». En: *Summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science* (2010).

[4] Jack Dongarra y col. «The International Exascale Software Project Roadmap». En: *International Journal of High Performance Computing Applications* 25.1 (2011), págs. 3-60.

[5] Marc Duranton y col. *The HIPEAC vision for advanced computing in horizon 2020*. 2013.

[6] S. H. Fuller y L. I. Millett. *The Future of Computing Performance: Game Over Or Next Level?* National Academy Press, 2011.

[7] Eugen Feller, Christine Morin y Daniel Leprince. *State of the Art of Power Saving in Clusters and Results from the EDF Case Study*. RR-7473. INRIA, 2010.

[8] Hui Chen, Shinan Wang y Weisong Shi. «Where does the power go in a computer system: Experimental analysis and implications». En: *2011 International Green Computing Conference and Workshops (IGCC)*. 2011, págs. 1-6.

[9] Steve Greenberg y col. «Best Practices for Data Centers: Lessons Learned from Benchmarking 22 Data Centers». En: *Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings in Asilomar, CA. ACEEE*. 2006, págs. 76-87.

[10] Urs Hoelzle y Luiz Andre Barroso. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 1ª ed. Morgan y Claypool Publishers, 2009.

[11] H.K. Pyla. *High Performance Thermal-aware Distributed Computing*. VDM Publishing, 2008.

[12] A. Vasan y col. «Worth their watts? - an empirical study of datacenter servers». En: *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*. 2010, págs. 1-10.

[13] Vinícius Garcia Pinto y col. «Energy Efficiency Evaluation of Multilevel Parallelism on Low Power Processors». En: *Anais do XXXIV Congresso da Sociedade Brasileira de Computação (WPerfomance - XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação)*. Porto Alegre, 2014, págs. 1825-1836.

[14] Frank G. Soltis. *Fortress Rochester: The Inside Story of the IBM iSeries*. 29th Street Press, 2001.

[15] The Green 500. *The Green500 List*. `http://www.green500.org`, 2015.

[16] NVIDIA Corporation. *NVIDIA CUDA API Reference Manual 6.0*. 2012.

[17] Khronos OpenCL Working Group. *OpenCL 2.0 Specification*. 2014.

[18] TOP500. *The Top500 List*. `http://www.top500.org`, 2015.

[19] The rCUDA Team. *rCUDA: Remote CUDA*. `http://www.hpca.uji.es/rcuda`, 2014.

[20] Pawankumar Hedge Katie Roberts-Hoffman. *ARM vs. Intel Atom: Architectural and Benchmark Comparisons*. University of Texas. 2009.

[21] Mateusz Jarus y col. «Performance Evaluation and Energy Efficiency of High-Density HPC Platforms Based on Intel, AMD and ARM Processors». En: *Energy Efficiency in Large Scale Distributed Systems*. Ed. por Jean-Marc Pierson, Georges Da Costa y Lars Dittmann. Vol. 8046. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, págs. 182-200.

[22] Rafael Vidal Aroca y Luiz Marcos Garcia Gonçalves. «Towards Green Data Centers: A Comparison of x86 and ARM Architectures Power Efficiency». En: *Journal of Parallel and Distributed Computing* 72.12 (2012), págs. 1770-1780.

[23] Phillip Stanley-Marbell y Victoria Caparrós Cabezas. «Performance, Power, and Thermal Analysis of Low-Power Processors for Scale-Out Systems.» En: *IPDPS Workshops*. IEEE, 2011, págs. 863-870.

[24] Nikilesh Balakrishnan. *Building and benchmarking a low power ARM cluster*. The University of Edinburgh, 2012.

[25] MichaelA. Laurenzano y col. «Characterizing the Performance-Energy Tradeoff of Small ARM Cores in HPC Computation». En: *Euro-Par 2014 Parallel Processing*. Ed. por Fernando Silva, Inês Dutra y Vítor Santos Costa. Vol. 8632. Lecture Notes in Computer Science. Springer International Publishing, 2014, págs. 124-137.

[26] Nikola Rajovic y col. «Tibidabo: Making the case for an ARM-based HPC system». En: *Future Generation Computer Systems* (2013).

[27] M. Duranton y col. *HiPEAC vision 2015*. `https://www.hipeac.net/assets/public/publications/vision/hipeac-vision-2015_Dq0boL8.pdf`, 2015.

[28] H. Esmaeilzadeh y col. «Dark silicon and the end of multicore scaling». En: *Proc. 38th Annual Int. Symp. on Computer architecture*. ISCA'11. 2011, págs. 365-376.

[29] R. Lucas. *Top ten Exascale research challenges*. DOE ASCAC Subcommittee Report. 2014.

[30] Barcelona Supercomputing Center. *The Mont Blanc Project*. `http://montblanc-project.eu`, 2014.

[31] ARM Ltd. *VFPv3 Floating Point Unit*. `http://www.arm.com/products/processors/technologies/vector-floating-point.php`, 2014.

[32] ARM Ltd. *The ARM NEON general purpose SIMD engine*. `http://www.arm.com/products/processors/technologies/neon.php`, 2014.

[33] J. Turley. *Cortex-A15 "Eagle" flies the coop*. The Linley Group. http://www.linleygroup.com/newsletters/newsletter_detail.php?num=3986, 2014.

[34] ARM Ltd. *Cortex-A Series*. http://www.arm.com/products/processors/cortex-a/index.php, 2014.

[35] NVIDIA Corporation. *Procesadores Tegra 3*. http://www.nvidia.es/object/tegra-3-es.html, 2014.

[36] NVIDIA Corporation. *Variable SMP – A Multi-Core CPU Architecture for Low Power and High Performance*. 2011.

[37] NVIDIA Corporation. *NVIDIA Tegra k1*. http://www.nvidia.es/object/tegra-k1-processor-es.html, 2015.

[38] Intel Corporation. *Intel Atom Processor*. http://www.intel.com/content/www/us/en/processors/atom/atom-processor.html, 2014.

[39] TimS. Woodall y col. «High Performance RDMA Protocols in HPC». En: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. por Bernd Mohr y col. Vol. 4192. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, págs. 76-85.

[40] Brice Goglin. «What HPC networking requires from the Linux Kernel». En: *HPCwire's Daily Coverage* (2006).

[41] Tarick Bedeir. *RDMA Read and Write with IB Verbs*. Schlumberger, 2010.

[42] Jia Song y Jim Alves-Foss. «Performance Review of Zero Copy Techniques». En: *International Journal of Computer Science and Security (IJCSS)* 6.4 (2012).

[43] Mellanox Technologies Inc. *RDMA Aware Networks Programming User Manual*. Mellanox Technologies Inc. 2013.

[44] Tarick Bedeir. *Basic Flow Control for RDMA Transfers*. Schlumberger, 2013.

[45] The OpenFabrics Alliance. *OFED Overview*. https://www.openfabrics.org/index.php/ofed-for-linux-ofed-for-windows/ofed-overview.html, 2014.

[46] Gregory Kerr. *Dissecting a Small InfiniBand Application Using the Verbs API*. Northeastern University, 2011.

[47] Tarick Bedeir. *Building an RDMA-Capable Application with IB Verbs*. Schlumberger, 2010.

[48] IBTA. *About Infiniband*. `http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband`, 2014.

[49] Mellanox Technologies Inc. *Introduction to InfiniBand*. 2006.

[50] Mellanox Technologies Inc. *Introducing EDR 100Gb/s - Enabling the Use of Data*. 2014.

[51] Oracle Corporation. *Delivering Application Performance with Oracle's InfiniBand Technology*. 2012.

[52] Chelsio Communications Inc. *LAMMPS and WRF on iWARP vs. InfiniBand FDR*. 2013.

[53] K. Kandalla y col. «Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters». En: *The 39th International Conference on Parallel Processing (ICPP - 2010)*. 2010, págs. 218-227.

[54] Charles R. Maule. «iWARP Ethernet: Key to Driving Ethernet into High Performance Environments». En: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. SC '06. Tampa, Florida: ACM, 2006.

[55] Mellanox Technologies Inc. *RoCE vs. iWARP Competitive Analysis Brief*. 2010.

[56] Robert Pan. «Ethernet versus Infiniband». En: *Canadian Young Scientist Journal* 2014.2 (2014), págs. 11-14.

[57] M.J. Rashti y A. Afsahi. «10-Gigabit iWARP Ethernet: Comparative Performance Analysis with InfiniBand and Myrinet-10G». En: *2007 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2007, págs. 1-8.

[58] Brian Hausauer. *iWARP Ethernet: Eliminating Overhead In Data Center Design*. NetEffect Inc, 2012.

[59] NetEffect Inc. *Understanding iWARP: Eliminating Overhead and Latency in multi-Gb Ethernet Networks*. 2012.

[60] F.D. Neeser, B. Metzler y P.W. Frey. «SoftRDMA: Implementing iWARP over TCP kernel sockets». En: *IBM Journal of Research and Development* 54.1 (2010), págs. 1-16.

[61] D. Dalessandro, A. Devulapalli y P. Wyckoff. «iWARP protocol kernel space software implementation». En: *20th International Parallel and Distributed Processing Symposium (IPDPS)*. 2006.

[62] Intel Corporation. *Understanding iWARP: Delivering Low Latency to Ethernet*. 2010.

[63] Michael Fenn; Lazaro Calderin; Jeffrey Nucciarone; Vikas Argod. *Evaluation of iWARP versus InfiniBand Performance*. Pennsylvania State University, 2011.

[64] J. Vienne y col. «Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems». En: *2012 IEEE 20th Annual Symposium on High-Performance Interconnects (HOTI)*. 2012, págs. 48-55.

[65] System Fabrics Works. *SoftRoCE*. `http://www.systemfabricworks.com/home`, 2014.

[66] David A. Patterson y John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. 3ª ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[67] Sergio Barrachina y col. «Exploiting the Capabilities of Modern GPUs for Dense Matrix Computations». En: *Concurrency and Computation: Practice & Experience* 21.18 (2009), págs. 2457-2477.

[68] J. Enos y col. «Quantifying the impact of GPUs on performance and energy efficiency in HPC clusters». En: *2010 International Green Computing Conference*. 2010, págs. 317-324.

[69] vCORE Inc. *NextIO*. Diciembre de 2013. `http://www.nextio.com`, 2014.

[70] Adrián Castelló y col. «On the Use of Remote GPUs and Low-Power Processors for the Acceleration of Scientific Applications». En: *ENERGY 2014, The Fourth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*. 2014, págs. 57-62.

[71] Adrián Castelló y col. «Acelerando Aplicaciones Científicas Con GPUs Remotas y Procesadores De Bajo Consumo». En: *XXV Jornadas de paralelismo*. 2014, págs. 187-193.

[72] Albano Alves y col. «dOpenCL - Supporting Distributed Heterogeneous Computing in HPC Clusters». En: *Euro-Par 2012: Parallel Processing Workshops*. Vol. 7640. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, págs. 112-122.

[73] A. Barak y col. «A package for OpenCL based heterogeneous computing on clusters with many GPU devices». En: *2010 IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*. 2010, págs. 1-7.

[74] Jungwon Kim y col. «SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters». En: *Proceedings of the 26th ACM International Conference on Supercomputing*. ICS '12. San Servolo Island, Venice, Italy: ACM, 2012, págs. 341-352.

[75] S. Xiao y col. «VOCL: An Optimized Environment for Transparent Virtualization of Graphics Processing Units». En: *1st Innovative Parallel Computing*. San Jose, CA: IEEE, 2012.

[76] NVIDIA Corporation. *NVIDIA CUDA Libraries*. 2012.

[77] Jason Sanders y Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[78] David B. Kirk y Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. 1ª ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[79] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide 6.0*. 2012.

[80] NVIDIA Corporation. *Popular GPU-Accelerated Applications Catalog*. NVIDIA Corporation. 2012.

[81] NVIDIA Corporation. *NVIDIA CUDA Samples 6.0*. 2012.

[82] Lin Shi, Hao Chen y Jianhua Sun. «vCUDA: GPU accelerated high performance computing in virtual machines». En: *2009 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. 2009, págs. 1-11.

[83] Zillians Inc. *V-GPU: GPU virtualization*. `http://www.zillians.com/vgpu`, 2013.

[84] Vishakha Gupta y col. «GViM: GPU-accelerated Virtual Machines». En: *3rd Workshop on System-level Virtualization for High Performance Computing*. Nuremburg, Germany: ACM, 2009, págs. 17-24.

[85] Giulio Giunta y col. «A GPGPU Transparent Virtualization Component for High Performance Computing Clouds». En: *Euro-Par 2010: Parallel Processing Workshops*. Vol. 6271. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, págs. 379-391.

[86] Minoru Oikawa y col. «DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment». En: *2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*. IEEE Computer Society, 2012, págs. 1207-1214.

[87] J. Duato y col. «rCUDA: Reducing the number of GPU-based accelerators in high performance clusters». En: *2010 International Conference on High Performance Computing and Simulation (HPCS)*. 2010, págs. 224-231.

[88] Carlos Reaño y col. «CU2rCU: Towards the complete rCUDA remote GPU virtualization and sharing solution». En: *2012 19th International Conference on High Performance Computing (HiPC)*. 2012, págs. 1-10.

[89] Jose Duato y col. «Performance of CUDA Virtualized Remote GPUs in High Performance Clusters». En: *Proceedings of the 2011 International Conference on Parallel Processing*. ICPP '11. Washington, DC, USA: IEEE Computer Society, 2011, págs. 365-374.

[90] José Duato y col. «Enabling CUDA acceleration within virtual machines using rCUDA». En: *Proceedings of the 2011 International Conference on High Performance Computing (HiPC 2011)*. Bangalore, India, 2011, págs. 1-10.

[91] Antonio J. Peña. «Virtualization of Accerlerators in High Performance Clusters». Universidad Jaume I - Castellón, 2012.

[92] José Duato y col. «An efficient implementation of GPU virtualization in high performance clusters». En: *Euro-Par 2009, Parallel Processing – Workshops*. Lecture Notes in Computer Science 6043 (2010), págs. 385-394.

[93] Hyun-Wook Jin y Chuck Yoo. «Analysis and Enhancement of Pipelining the Protocol Overheads for a High Throughput.» En: *PDPTA*. Ed. por Hamid R. Arabnia y Youngsong Mun. CSREA Press, 2003, págs. 935-941.

[94] Carlos Reaño y col. «Influence of InfiniBand FDR on the performance of remote GPU virtualization». En: *IEEE Cluster 2013*. Indianapolis, IN, 2013.

[95] NVIDIA Corporation. *NVIDIA GPUDirect Technology*. `http://developer.nvidia.com/gpudirect`, 2011.

[96] H. Wang y col. «GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation». En: *IEEE Transactions on Parallel and Distributed Systems* 99 (2013).

[97] Hiroshi Tezuka y col. «Pin-Down Cache: A Virtual Memory Management Technique for Zero-Copy Communication.» En: *IPPS/SPDP*. 1998, págs. 308-314.

[98] Jiesheng Wu, Pete Wyckoff y Dhabaleswar K. Panda. «Supporting Efficient Noncontiguous Access in PVFS over InfiniBand.» En: *CLUSTER*. IEEE Computer Society, 2003, págs. 344-.

[99] Li Ou y Jizhong Han. «A Fast Read/Write Process to Reduce RDMA Communication Latency.» En: *IWNAS*. IEEE Computer Society, 2006, págs. 138-144.

[100] Electronic Education Devices. *Watts Up? Plug Load Meters*. `https://www.wattsupmeters.com/secure/products.php?pn=0`, 2014.

[101] Chung-Hsing Hsu y S.W. Poole. «Power measurement for high performance computing: State of the art». En: *2011 International Green Computing Conference and Workshops (IGCC)*. 2011, págs. 1-6.

[102] Susanne Albers. «Energy-Efficient Algorithms». En: *Communications of the ACM* 53.5 (2010), págs. 86-96.

[103] NVIDIA Corporation. *The Benefits of Multiple CPU Cores in Mobile Devices*. 2010.

[104] Adrián Castelló. «Virtualización de GPUs en Procesadores de Bajo Consumo». Universitat Jaume I, 2014.

[105] D. Turner y Xuehua Chen. «Protocol-dependent message-passing performance on Linux clusters». En: *2002 IEEE International Conference on Cluster Computing*. 2002, págs. 187-194.

[106] Dave Turner y col. «Integrating New Capabilities into NetPIPE». En: *PVM/MPI*. Ed. por Jack Dongarra, Domenico Laforenza y Salvatore Orlando. Vol. 2840. Lecture Notes in Computer Science. Springer, 2003, págs. 37-44.

[107] OFED. *OpenFabrics Software Components*. `https://www.openfabrics.org/downloads/perftest`, 2014.

[108] R. Ge y col. *Power Measurement Tutorial for the Green500 List*. 2007. `http://www.green500.org/sites/default/files/tutorial.pdf`, 2013.

[109] The ACPI Promoters Corporation. *ACPI - Advanced Configuration & Power Interface*. `http://www.acpi.info`, 2013.

[110] V.M. Weaver y col. «Measuring Energy and Power with PAPI». En: *41st International Conference on Parallel Processing Workshops (ICPPW)*. 2012, págs. 262-268.

[111] Jun De Vega. *Intel Power Gadget*. `https://software.intel.com/en-us/articles/intel-power-gadget`, 2014.

[112] Thanh Do, Suhib Rawshdeh y Weisong Shi. «ptop: A process-level power profiling tool». En: *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower09)*. 2009.

[113] W.L. Bircher y L.K. John. «Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events». En: *2007 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*. 2007, págs. 158-168.

[114] Rong Ge y col. «PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications». En: *IEEE Transactions on Parallel and Distributed Systems* 21.5 (2010), págs. 658-671.

[115] S. Ryffel. «The Linux energy attribution and accounting platform». Swiss Federal Inst. of Technology, 2009.

[116] D. Bedard y col. «PowerMon: Fine-grained and integrated power monitoring for commodity computer systems». En: *Proceedings of the IEEE SoutheastCon*. 2010, págs. 479-484.

[117] Intel Corporation. *Intel Energy Checker SDK*. `http://software.intel.com/enus/articles/intel-energy-checker-sdk`, 2013.

[118] SPEC. *SPEC Power and Performance Benchmark Methodology*. 2011.

[119] J. Stanley, K. Brill y Koomey J. *Four Metrics Define Data Center 'Greenness': Enabling Users to Quantify Energy Consumption Initiatives for Environmental Sustainability and "Bottom Line" Profitability*. Uptime Institute Inc., 2007.

[120] Balaji Subramaniam y Wu-chun Feng. «The Green Index: A Metric for Evaluating System-Wide Energy Efficiency in HPC Systems». En: *8th IEEE Workshop on High-Performance, Power-Aware Computing (HPPAC)*. Shanghai, China, 2012.

[121] Timo Minartz, JulianM. Kunkel y Thomas Ludwig. «Simulation of power consumption of energy efficient cluster hardware». En: *Computer Science - Research and Development* 25.3-4 (2010), págs. 165-175.

[122] Arndt Bode. «Energy to Solution: A New Mission for Parallel Computing». En: *Proceedings of the 19th International Conference on Parallel Processing*. Euro-Par'13. Aachen, Germany: Springer-Verlag, 2013, págs. 1-2.

[123] Costas Bekas y Alessandro Curioni. «A new energy aware performance metric». En: *Computer Science - Research and Development* 25.3-4 (2010), págs. 187-195.

[124] Massimiliano Fatica. «Accelerating Linpack with CUDA on Heterogenous Clusters». En: *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. GPGPU-2. Washington, D.C., USA: ACM, 2009, págs. 46-51.

[125] J. Dongarra y col. *LINPACK Users Guide*. SIAM. Philadelphia, 1979.

[126] A. Petitet y col. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. 2008.

[127] Jakub Kurzak, Stanimire Tomov y Jack Dongarra. *Autotuning GEMMs for Fermi*. 2011.

[128] ATLAS Project. *Automatically Tuned Linear Algebra Software (ATLAS)*. http://math-atlas.sourceforge.net, 2013.

[129] Sandia National Laboratories. *LAMMPS Molecular Dynamics Simulator*. Sandia National Labs. http://lammps.sandia.gov, 2014.

[130] Intel Corporation. *Intel Ethernet 10-Gigabit iWARP LAMMPS Performance Study*. 2012.

[131] Chelsio Communications Inc. *LAMMPS, LS-DYNA, HPL and WRF on iWARP vs. InfiniBand FDR*. 2013.

[132] MIT Laboratory for Computer Science. *FFTW - Discrete Fourier Transform (DFT)*. http://www.fftw.org, 2013.

[133] NVIDIA Corporation. *NVIDIA Jetson TK1 Development Kit Bringing GPU-accelerated computing to Embedded Systems*. 2014.

[134] NVIDIA Corporation. *Meet CARMA - The CUDA on ARM Development Kit*. https://research.nvidia.com/news/meet-carma-cuda-arm-development-kit, 2014.

[135] Yongchao Liu. *CUDASW++ - GPU accelerated Smith-Waterman algorithm*. http://cudasw.sourceforge.net/homepage.htm, 2014.

[136] Doug Hains y col. «Improving CUDASW++, a Parallelization of Smith-Waterman for CUDA Enabled Devices.» En: *IPDPS Workshops*. IEEE, 2011, págs. 490-501.

[137] Yongchao Liu, Bertil Schmidt y Douglas Maskell. «CUDASW++ 2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions». En: *BMC Research Notes* 3.1 (2010), págs. 93-102.

[138] Yongchao Liu, Adrianto Wirawan y Bertil Schmidt. «CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions». En: *BMC Bioinformatics* 14.1 (2013), págs. 1-10.

[139] Yongchao Liu, Douglas L. Maskell y Bertil Schmidt. «CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units.» En: *BMC research notes* 2.1 (2009), págs. 73-81.

[140] Liu Yongchao. *CUDASW++ query sequences*. Octubre de 2013. http://sourceforge.net/projects/cudasw/files/data,

[141] UniProt. *Latest Swiss-Prot database*. ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.fasta.gz, 2013.

[142] Carnegie Mellon University. *GPU-Blast*. http://archimedes.cheme.cmu.edu/?q=gpublast, 2014.

[143] Panagiotis D. Vouzis y Nikolaos V. Sahinidis. «GPU-BLAST: using graphics processors to accelerate protein sequence alignment». En: *Bioinformatics* 27.2 (2011), págs. 182-188.

[144] M. Castillo y col. «Analysis of Strategies to Save Energy for Message-Passing Dense Linear Algebra Kernels». En: *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2012, págs. 346-352.