Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Estudio analítico de las prestaciones de una aplicación paralela distribuida

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Antonio Díaz Román

**Tutores**: Federico Silla Jiménez

Javier Prades Gasulla

2015/2016

# Resumen

La ejecución de aplicaciones con altas cargas computacionales adquiere una creciente demanda hoy en día, por lo que se manifiesta la necesidad de contar con una infraestructura adecuada y una correcta gestión de la misma por parte de su responsable. En este trabajo se realiza una simulación de la administración de un clúster de altas prestaciones que ejecuta aplicaciones paralelas distribuidas, poniendo especial atención a los métodos para llevar a cabo la ejecución y a los resultados de rendimiento obtenidos, con objeto de adquirir la experiencia necesaria para administrar este tipo de sistemas y por consiguiente proporcionar una administración apropiada a los usuarios de este tipo de aplicaciones.

**Palabras clave:** GPU, MPI, prestaciones, sistemas distribuidos, redes de computadores

# Abstract

Running applications with high computational loads acquires a growing demand today, so the need to adequate infrastructure and proper management of it by its responsible is manifested. This document describes a simulation of the administration of a high performance cluster running parallel distributed applications, paying special attention to the methods to carry out the implementation and to the performance results obtained, in order to acquire the necessary experience to manage this kind of systems and therefore provide an appropriate administration to users of this kind of applications.

**Keywords:** GPU, MPI, performance, distributed systems, computer networks



# Tabla de contenidos

# 1. Introducción

El plan de estudios de la Escuela Técnica Superior de Ingeniería Informática (ETSINF) de la Universidad Politécnica de Valencia (UPV) es suficientemente amplio como para proporcionar a sus titulados la diversidad de conocimientos necesarios para poder desarrollar su posterior carrera profesional en cualquiera de las áreas relacionadas con la informática. Obviamente, y dadas las limitaciones temporales del plan de estudios, que debe comprimir toda la formación en cuatro años, la inmensa mayoría de ingenieros egresados de la ETSINF tendría que completar su formación con conocimientos específicos del área en el que desarrollan su profesión. Es posible incluso que a lo largo de la vida profesional del ingeniero, éste vaya cambiando de área con el tiempo. También es probable, dada la naturaleza cambiante de la informática, que aunque el egresado ejerza su profesión siempre en el mismo área, éste tenga que refrescar sus conocimientos para incorporar los últimos desarrollos y avances.

Las áreas en las que un ingeniero de la ETSINF puede desarrollar su vida profesional son muy diversas. Por ejemplo, puede trabajar en las áreas de la ingeniería del software, la inteligencia artificial y robótica, la ingeniería de computadores, etc.

Otra de las áreas en las que un ingeniero de la ETSINF puede desarrollar su carrera profesional es trabajando como administrador de sistemas (comúnmente conocido como *admin* en la jerga del mundillo informático). Las labores de un administrador de sistemas consisten en implementar, configurar, mantener, monitorizar, documentar y asegurar el correcto funcionamiento de un sistema informático, o algún aspecto de éste.

Los conocimientos que necesita un administrador de sistemas han sido parcialmente cubiertos por el plan de estudios de la ETSINF. De esta manera, asignaturas como “Estructura de Computadores”, “Arquitectura e Ingeniería de Computadores”, “Redes de Computadores”, “Fundamentos de Sistemas operativos”, “Concurrencia y Sistemas Distribuidos” y “Bases de Datos y Sistemas de Información”, entre otras, cubren holgadamente buena parte del contexto en el que se enmarca el trabajo de un administrador de sistemas. No se debe dejar de mencionar la asignatura “Diseño, Configuración y Evaluación de Sistemas Informáticos”, donde se revisan conceptos relacionados con los factores de diseño de sistemas informáticos, análisis comparativos de rendimiento, monitorización de programas y sistemas, tecnologías de almacenamiento y consumo energético de los sistemas informáticos, por ejemplo.

Un administrador de sistemas puede ejercer su carrera profesional en diversos entornos: desde un centro de datos de una empresa donde se manejen principalmente bases de datos hasta sistemas cloud donde se dé servicio a usuarios externos mediante el uso de máquinas virtuales, es decir, Infrastructure as a Service (IaaS), o mediante otros modelos como Platform as a Service (Paas) o Software as a Service (SaaS). También es posible ejercer de administrador de sistemas en un centro de computación de altas prestaciones. De hecho, en la actualidad, la computación de altas prestaciones está experimentando una creciente demanda en múltiples campos, entre los que destacan el análisis predictivo (por ejemplo, el usado en el mundo de las finanzas al calcular posibles tendencias futuras del mercado), la simulación (como la usada en astrofísica al simular el comportamiento de galaxias con miles de millones de estrellas), la gestión de grandes volúmenes de datos (es decir, el llamado Big Data o datos masivos) y la investigación en general, entre otros. Por esta razón, la implantación de sistemas informáticos que puedan ejecutar intensas cargas de trabajo en el menor tiempo posible está siendo enormemente impulsada por numerosas empresas privadas y organismos públicos.

Sin embargo, la administración de estos computadores de altas prestaciones no resulta una tarea sencilla: necesita de profesionales altamente cualificados que comprendan estos sistemas en todos sus niveles. En consecuencia, y con una correcta administración del sistema por parte del responsable del mismo, los usuarios de estos sistemas pueden resultar altamente beneficiados recibiendo los recursos precisos de una manera apropiada durante el tiempo que se estime necesario.

Para ejercer de administrador de sistemas en un centro de computación de altas prestaciones, un ingeniero titulado por la ETSINF de la UPV ha adquirido ciertos conocimientos que pasan por haber estudiado materias relacionadas con la computación de altas prestaciones. En particular, los conocimientos base son los que determina el plan de estudios del Grado en Ingeniería Informática de la UPV, el cual dispone de un gran abanico de asignaturas que cubren parcialmente el contexto de la computación de altas prestaciones, como por ejemplo, las asignaturas “Computación Paralela”, “Arquitectura e Ingeniería de Computadores” y “Diseño, Configuración y Evaluación de Sistemas Informáticos”. No obstante, los conocimientos adquiridos durante el estudio del Grado deben ser complementados, bien porque es necesario profundizar en algunos de ellos, bien porque es necesario adquirir otros nuevos. Por ejemplo, y sin ir más lejos, en el contexto de la computación de altas prestaciones es extremadamente habitual el uso de aceleradores externos y librerías asociadas que no son cubiertas por ninguna asignatura del Grado, lo que supone un claro ejemplo de materia en la que convendría profundizar. A todo esto, como materia nueva en la que habría que formarse y que no tiene apenas

presencia obligatoria en el plan de estudios del Grado, es la creación de aplicaciones gráficas y multimedia. A un administrador de sistemas podría interesarle conocer en cierta medida aspectos fundamentales en estos campos, pues en ocasiones la creación de contenido multimedia necesita de unos recursos especiales, como podría ser un computador de altas prestaciones para el renderizado de secuencias de vídeo.

Dado el interés que un trabajo de administración de sistemas puede tener para un ingeniero en informática, en este Trabajo Fin de Grado se propone al alumno profundizar en esta área. De esta forma, se propone al alumno realizar una simulación de algunas de las funciones que llevaría a cabo si ejerciera su profesión como administrador de sistemas. En concreto, el objetivo de este Trabajo Fin de Grado consiste en la implantación y estudio de la ejecución de una aplicación paralela distribuida en un clúster de altas prestaciones. Con ello se pretende conocer en profundidad las métricas de los resultados de la ejecución de aplicaciones paralelas distribuidas cuando se ejecutan en un sistema de altas prestaciones, con objeto de administrar y gestionar correctamente los recursos del sistema en el momento de ejecutar este tipo de aplicaciones en un entorno multiusuario.

La tarea principal en este trabajo es la instalación y evaluación de prestaciones de una aplicación paralela distribuida. El principal motivo para ello es que un futuro administrador de sistemas debería conocer en qué condiciones y con qué configuración se obtiene el mejor rendimiento en una aplicación de este tipo, debido a que sus usuarios no tienen razón alguna para poseer esos conocimientos, ya que no necesariamente han estudiado materias relacionadas con la informática. Así pues, el usuario de Gromacs espera que el administrador del sistema encargado de administrar y gestionar el clúster conozca la mejor configuración para ejecutar sus simulaciones.

También se persigue poner en contacto al alumno con diversas piezas de tecnología punta, habituales en numerosos centros de computación de altas prestaciones, como pueden ser GPUs de última generación o los últimos desarrollos hardware de altas prestaciones.

Dada la naturaleza del presente trabajo, resultaría muy interesante el análisis de un número variable de aplicaciones paralelas distribuidas, con objeto de alcanzar una visión más amplia sobre el comportamiento de este tipo de aplicaciones y poder extrapolar los resultados tanto a las aplicaciones futuras como a las aplicaciones que no son objeto de estudio. No obstante, el esfuerzo requerido para llevar a cabo tal cometido excedería con mucho el del objetivo principal de este Trabajo Fin de Grado, por lo que el alcance del mismo se limita al estudio de una aplicación que puede ser ejecutada

utilizando la gran mayoría de las tecnologías actuales en el ámbito de la computación de altas prestaciones.

Y dado el objetivo de este trabajo, los principales recursos de los que se hace uso para la elaboración del mismo son la aplicación Gromacs y el clúster de altas prestaciones VirtualGAP, ubicado en la sala de ordenadores de la planta baja del edificio 1G de la Escuela Técnica Superior de Ingeniería Informática de la UPV. Este clúster cuenta con diversos tipos de nodos, entre los que destacan los nodos Supermicro 7048, de 4 U y 4 GPUS, y nodos Supermicro 1027, de 1 U y 1 GPU. Ambos tipos cuentan con microprocesadores Intel Xeon e5 2620 v2, con una frecuencia de funcionamiento de 2,1 GHz y 15 MB de caché. El clúster también cuenta con una red InfiniBand en sus versiones FDR (56 Gbps) y EDR (100 Gbps).

El trabajo realizado en este Trabajo Fin de Grado ha consistido en:

- Un estudio previo de la documentación de Gromacs y de conceptos relacionados con la computación de altas prestaciones usando aceleradores externos.
- Instalación de Gromacs en el clúster VirtualGAP dando soporte a las versiones CPU y GPU.
- Lanzar un determinado número de ejecuciones de Gromacs en todas sus variantes, modificando los parámetros de entrada para obtener distintos resultados.
- Tomar los datos obtenidos en todas las ejecuciones realizadas, incluyendo medidas de prestaciones temporales y consumo de energía con la finalidad de encontrar cual es la mejor configuración hardware/software para la ejecución de esta aplicación, con la finalidad de proporcionar a los usuarios del centro de datos dicha información.
- Realizar un análisis comparativo de los resultados obtenidos.

Finalmente, este trabajo se estructura principalmente en cinco secciones bien diferenciadas. En primer lugar, se realizará una breve descripción de los principales conceptos relevantes que se han manejado con asiduidad.

Seguidamente, se realizará una descripción de las distintas instalaciones de Gromacs efectuadas en el clúster de alto rendimiento VirtualGAP con objeto de poder llevar a cabo su ejecución en todas sus variantes.

En tercer lugar, se efectuará una breve descripción de los tipos de ejecución que se han realizado de Gromacs, prestando especial atención a los parámetros establecidos para su ejecución.

A continuación, se expondrán los resultados de rendimiento en la ejecución de Gromacs en sus distintas modalidades. Se pondrá especial atención a la métrica de rendimiento de Gromacs en cada una de las ejecuciones de la aplicación en base a los parámetros recibidos en su lanzamiento.

Y para terminar, se presentarán las conclusiones de la simulación a la que se ha sometido al alumno en este Trabajo Fin de Grado, con objeto de introducirle en la administración de un clúster de altas prestaciones que ejecuta aplicaciones paralelas distribuidas.

# 2. Conceptos previos

El propósito de este capítulo es dar a conocer algunos conceptos que no han sido cubiertos parcial o totalmente en el plan de estudios del Grado en Ingeniería Informática de la UPV y que, sin embargo, han sido conceptos clave para el desarrollo de este trabajo, ya que en su mayoría se trata de tecnologías que se utilizan intensamente en la computación de altas prestaciones.

## 2.1. Gromacs

Gromacs [1] es una aplicación de alto rendimiento, de código abierto y multiplataforma que se utiliza para la simulación de la dinámica molecular de sistemas con cientos de millones de partículas, y es capaz de simular proteínas, lípidos y ácidos nucleicos. Fue desarrollado inicialmente por la Universidad de Groningen, en Países Bajos, si bien actualmente es mantenido por contribuidores de universidades y centros de investigación de todo el mundo. Gromacs es gratuito y libre bajo la Licencia Pública General (GPL).

Al igual que el resto de aplicaciones de alto rendimiento, Gromacs tiene la capacidad de ejecutarse tanto en plataformas formadas por un único nodo como en sistemas con múltiples nodos (clúster). Es por ello que en este trabajo se ejecutará Gromacs en el clúster VirtualGAP, donde existirá la posibilidad de ejecutarlo tanto en un único nodo del clúster como en varios de ellos.

Por otra parte, los cálculos realizados durante la ejecución de esta aplicación pueden llevarse a cabo en CPUs tradicionales o GPU (véase punto 2.2).

## 2.2.GPU

Una GPU o Graphics Processing Unit [2] es un microprocesador especializado que se encuentra en las tarjetas gráficas cuyo objetivo principal es el procesamiento de gráficos y las operaciones de coma flotante. Debido a su naturaleza para procesar gráficos, su potencia para ejecutar paralelismo y operaciones en coma flotante es

sensiblemente mayor que la de los microprocesadores de propósito general, por lo que se usan ampliamente en computación de altas prestaciones, aunque en este contexto se suele hacer alusión a las GPUs como GPGPUs, es decir, GPUs de propósito general.

Para programar aplicaciones que puedan explotar las ventajas de las GPUs en el contexto de la paralelización se utiliza la librería CUDA [3], desarrollada por Nvidia para este propósito. Así pues, CUDA es un conjunto de herramientas que permite a los desarrolladores realizar programas que utilicen tarjetas gráficas Nvidia. De esta forma, los programas que hagan uso de CUDA pueden explotar las ventajas de las GPU frente a las CPU de propósito general mediante el uso del paralelismo que ofrecen sus múltiples núcleos, creando y ejecutando de forma simultánea un elevado número de hilos de ejecución.

Por otra parte, cuando se habla de CUDA es muy habitual escuchar el concepto de NVML [4], que es una interfaz de programación de aplicaciones propietaria de nVidia que posibilita la monitorización y gestión directa de las GPUs nVidia. En este Trabajo Fin de Grado la interfaz NVML será usada para poder modificar dinámicamente los relojes de la GPU con objeto de mejorar ligeramente el rendimiento.

Por otra parte, también es posible programar GPU usando OpenCL. OpenCL [5] es un estándar que permite la programación de aplicaciones usando paralelismo a nivel de datos y de tareas que puede ejecutarse tanto en microprocesadores de propósito general como en GPUs. La especificación original es de Apple y fue desarrollada conjuntamente por AMD, IBM, Intel y Nvidia.

## 2.3. InfiniBand

InfiniBand [6] es un bus de comunicaciones serie de alta velocidad y baja latencia empleado ampliamente en clústeres de computadores para la comunicación entre los distintos nodos del clúster. Es el resultado de la fusión de dos diseños competidores, Future I/O y Next Generation I/O, desarrollados por diversas empresas tales como IBM, HP e Intel, aunque en la actualidad el único fabricante de esta tecnología es Mellanox.

Existen distintas variantes de InfiniBand, entre las que destacamos FDR y EDR, que ofrecen respectivamente un ancho de banda nominal de 56 Gbps y 100 Gbps. Se está trabajando ya en la siguiente generación, HDR, que ofrecerá un ancho de banda de 200 Gbps.

## 2.4. MPI

MPI [7] es un estándar que define la sintaxis y la semántica de las funciones contenidas en librerías desarrolladas para explotar la tecnología del paso de mensajes en el contexto de la ejecución de programas concurrentes en múltiples nodos de un clúster. El paso de mensajes se utiliza en programación concurrente, donde varios computadores trabajan en paralelo y acceden simultáneamente a los mismos datos.

La principal característica de MPI es que no requiere memoria compartida para funcionar, por lo que dada esta característica se utiliza ampliamente en sistemas distribuidos. Existen distintas librerías de paso de mensajes que implementan el estándar MPI, entre las que destacan OpenMPI [8] y MVAPICH2 [9].

## 2.5. Thread-MPI

Thread-MPI [10] es una implementación del estándar MPI basada en threads que incorpora Gromacs y que permite el uso del estándar MPI dentro de un único nodo sin ejecutar MPI directamente. Se trata del tipo por defecto de paralelización de Gromacs desde la versión 4.5 cuando se ejecuta en memoria compartida y, si no se indica lo contrario, Gromacs ejecuta tantos threads-MPI como núcleos lógicos tiene el nodo en cuestión. Puede combinarse con el uso de OpenMP e incluso puede deshabilitarse en favor de este último.

Lo habitual cuando se habla de paralelización en memoria compartida es hablar de OpenMP. De hecho, así es en la inmensa mayoría de escenarios donde se usa la paralelización en memoria compartida. Por ello, puede resultar confuso ver el concepto de MPI en memoria compartida, más todavía ver el concepto de thread en relación con MPI. Pero en el contexto de Gromacs, que usa Thread-MPI como paralelización por defecto en memoria compartida, es habitual mezclar los conceptos de MPI e hilos de ejecución en el ámbito de la ejecución de esta aplicación en un único nodo.

# 3. Instalación de Gromacs

En este capítulo se describirán los distintos tipos de instalaciones de Gromacs efectuadas en el clúster VirtualGAP, con objeto de dar soporte a los distintos tipos de ejecución que la aplicación admite. A grandes rasgos, la aplicación se ha instalado usando dos variantes:

- Memoria compartida
- Memoria distribuida usando MPI

Para el uso de la librería MPI se han considerado dos posibilidades diferentes: MVAPICH2 y OpenMPI. Asimismo, cada una de las modalidades anteriores tiene a su vez otros tres subtipos en función de la ausencia o presencia de GPUs en su ejecución:

- Ausencia de GPUs
- Presencia de GPUs
- Presencia de GPUs con soporte para NVML

La versión más reciente de Gromacs al tiempo de la elaboración del presente trabajo es la 5.1, y su código fuente puede obtenerse directamente de su web oficial www.gromacs.org. Cabe mencionar que un único paquete con el código fuente da soporte a todas las variantes mencionadas más arriba (memoria compartida o distribuida con y sin GPUs). Por otra parte, los desarrolladores de Gromacs han puesto a disposición del público unos tests que verifican la integridad de la compilación de la aplicación una vez dicha compilación ha concluido.

Tal y como se mencionó en el capítulo 1, se va a hacer uso del clúster VirtualGAP para la instalación y ejecución de Gromacs. Este clúster está situado en la sala de ordenadores de la planta baja del edificio 1G de la Escuela. Dicha sala se mantiene refrigerada a 19 grados centígrados y no dispone de espacio para puestos de trabajo, por lo que el acceso al clúster debe hacerse en remoto mediante conexiones tanto desde dentro de la Universidad Politécnica de Valencia como desde fuera de ella empleando una VPN (Virtual Private Network). La conexión empleada ha sido una Secure Shell (SSH) con autenticación mediante certificados. El clúster VirtualGAP dispone de unidades de red montadas mediante el protocolo NFS (Network File System) accesibles desde todos los nodos y que implementan dos características muy convenientes para el cumplimiento del objetivo propuesto para este TFG: la práctica totalidad de las librerías empleadas en este trabajo, que se ubican en las unidades de red, y el directorio *home*

donde se ubican los directorios personales de los usuarios del clúster, que también está ubicado en dichas unidades de red. El directorio *home* es donde ha sido instalada la aplicación Gromacs en todas sus variantes.

Para la instalación de Gromacs se ha recurrido a la web oficial [11] donde existe un amplio manual que permite la instalación usando múltiples opciones. La instalación no resulta extremadamente compleja siempre y cuando el usuario que la realiza tenga ciertas nociones básicas sobre el manejo de la shell Unix y se hayan obtenido previamente las librerías necesarias.

Así pues, una vez realizada la conexión con el clúster, la primera tarea sería la descarga del código fuente de Gromacs:

```
$ wget ftp://ftp.gromacs.org/pub/gromacs/gromacs-5.1.tar.gz
```

Y en segundo lugar, la obtención de los tests de verificación de integridad:

```
$ wget http://gerrit.gromacs.org/download/regressiontests-5.1.tar.gz
```

A continuación, se descomprimirían ambos paquetes:

```
$ tar xzf gromacs-5.1.tar.gz

$ tar xzf regressiontests-5.1.tar.gz
```

Y se generaría el directorio donde se volcará el resultado de la compilación:

```
$ cd gromacs-5.1

$ mkdir build

$ cd build
```

Lo visto hasta este punto constituye la parte común a todas las instalaciones realizadas, por lo que a partir de este momento cada instalación necesitará de unos parámetros exclusivos que se detallan a continuación.

## 3.1. Memoria compartida

Este tipo de instalación destaca por no dar soporte a ningún tipo de aceleración mediante unidades de procesamiento gráfico. Así pues, el primer paso para instalar esta variante de Gromacs sería construir el código fuente mediante la herramienta *cmake*:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-
compartida/source-code/regressiontests-5.1.2 -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-compartida/gromacs-
installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_GPU=OFF
```

Donde la opción -DREGRESSIONTEST_PATH determina la ruta del directorio donde se ha descomprimido el test de verificación, la opción DCMAKE_INSTALL_PREFIX le indica al compilador dónde debe instalar la aplicación una vez termine la compilación, la opción DGMX_FFT_LIBRARY establece la librería que implementa el algoritmo de la Transformada Rápida de Fourier, la opción DCMAKE_PREFIX_PATH indica una ruta no estándar para buscar librerías (y por tanto donde buscará la librería FFTW), y donde la opción DGMX_GPU=OFF determina que no se compilará Gromacs con opción a usar GPU. Gromacs realiza un uso muy intensivo de la librería FFTW, pero como es habitual en estos entornos no se incluye por defecto, de forma que se le proporciona al usuario la opción de usar otra distinta.

A continuación, y una vez construido el código fuente con *cmake*, se procede a la compilación propiamente dicha con *make*, aprovechando la capacidad para ejecutar paralelismo de los nodos del clúster VirtualGAP:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)
```

Donde `$(cat /proc/cpuinfo | grep -i processor | wc -l)` devuelve el número de núcleos lógicos de los que dispone el nodo desde donde se realiza la compilación.

Una vez realizada la compilación, se procede a comprobar la integridad de la misma mediante los test descargados previamente:

```
$ make check
```

Y por último, se procede con la instalación de Gromacs en el directorio indicado a *cmake*:

```
$ make install
```

Con esto, se obtendría una instalación limpia de Gromacs sin soporte a GPU en el directorio establecido.

## 3.2. Memoria compartida con GPU

En este tipo de instalación, la idea principal es dotar a Gromacs con el soporte para utilizar las GPUs instaladas en el nodo donde se ejecute la aplicación. De esta forma, durante la ejecución de Gromacs los procesos utilizarán las GPUs, que serán gestionadas mediante los threads OpenMP, que a su vez se ejecutarán en los núcleos del procesador de propósito general.

Para instalar la aplicación Gromacs con soporte para GPU se comienza, al igual que con el caso anterior, con la construcción del código fuente mediante *cmake*:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-compartida-GPU/source-code/regressiontests-5.1.2/ -DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-compartida-GPU/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_GPU=ON -DCUDA_TOOLKIT_ROOT_DIR=/nfs/LIBS/LIBS/CUDA/CURRENT/
```

Donde el parámetro DGMX_GPU=ON indica que se habilitará el soporte para utilizar GPU, y el parámetro DCUDA_TOOLKIT_ROOT_DIR indica la ruta donde se encuentra la librería CUDA. Por otra parte, nótese que el parámetro DREGRESSIONTEST_PATH es diferente al caso de memoria compartida. El motivo es que en este TFG vamos a generar los binarios de cada una de las variantes mencionadas al principio del capítulo en un directorio diferente para poder usar después todos ellos en las pruebas posteriores.

A continuación, se realizarían los mismos pasos que en el caso de compilar Gromacs sin soporte a GPU:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.3. Memoria compartida con GPU y NVML

En este caso, la única diferencia con respecto a la variante inmediatamente anterior es el uso de NVML, esto es, la librería de nVidia que permite ajustar dinámicamente los relojes de la GPU con objeto de mejorar el rendimiento final.

Por lo tanto, y al igual que en los casos anteriores, se procede con la construcción del código:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-compartida-
GPU-NVML/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-compartida-GPU-
NVML/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_GPU=ON -
DCUDA_TOOLKIT_ROOT_DIR=/nfs/LIBS/LIBS/CUDA/CURRENT/ -
DNVML_LIBRARY=/usr/lib64/libnvidia-ml.so -
DNVML_INCLUDE_DIR=/nfs/LIBS/LIBS/CUDA_GDK/352.79/usr/include/nvidia/gdk
```

Donde el parámetro DNVML_LIBRARY determina la ruta donde se encuentra el archivo principal de la librería NVML y el parámetro DNVML_INCLUDE_DIR especifica la ruta donde se encuentra el resto de archivos de la librería.

Se prosigue pues con la compilación e instalación de Gromacs de la forma habitual:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.4. Memoria distribuida con MVAPICH2

La instalación de Gromacs con soporte para memoria distribuida no difiere en gran medida de la instalación con memoria compartida. La diferencia relevante es la necesidad de indicar el soporte para MPI y declarar las rutas de los nuevos compiladores.



Así pues, la instalación comienza con la ejecución de *cmake* y los parámetros adecuados:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
MVAPICH2/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
MVAPICH2/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_GPU=OFF -
DGMX_MPI=ON -DMPI_CXX_COMPILER=/nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpicxx -
DMPI_C_COMPILER=/nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpicc
```

Donde destacan las opciones de DGMX_MPI=ON, que determina la intención de habilitar el soporte para memoria distribuida, y las opciones de DMPI_CXX_COMPILER y DMPI_C_COMPILER, que indican las rutas a los nuevos compiladores.

A continuación, el proceso finalizaría con la compilación, la comprobación de integridad y la instalación de Gromacs de la manera habitual:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.5. Memoria distribuida con MVAPICH2 y GPU

En este caso, la diferencia con respecto al anterior es, de nuevo, indicar al compilador la intención de habilitar el soporte para usar una GPU e indicar la ruta a las librerías CUDA:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
MVAPICH2-GPU/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-distribuida-MVAPICH2-
GPU/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_MPI=ON -
DMPI_CXX_COMPILER=/nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpicxx -
DMPI_C_COMPILER=/nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpicc -DGMX_GPU=ON -
DCUDA_TOOLKIT_ROOT_DIR=/nfs/LIBS/LIBS/CUDA/CURRENT/
```

Donde destacan, una vez más, las opciones de DGMX_GPU=ON para habilitar el soporte para GPU, y DCUDA_TOOLKIT_ROOT_DIR, para indicar la ruta a las librerías CUDA. El resto de tareas son idénticas al resto de casos:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.6. Memoria distribuida con MVAPICH2, GPU y NVML

El procedimiento para instalar esta variante de Gromacs se asemeja en gran medida al resto de casos, por lo que inicialmente se procede con *cmake*:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
MVAPICH2-GPU-NVML/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-distribuida-MVAPICH2-
GPU-NVML/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_MPI=ON -
DMPI_CXX_COMPILER=/nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpicxx -
DMPI_C_COMPILER=/nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpicc -DGMX_GPU=ON -
DCUDA_TOOLKIT_ROOT_DIR=/nfs/LIBS/LIBS/CUDA/CURRENT/ -
DNVML_LIBRARY=/usr/lib64/libnvidia-ml.so -
DNVML_INCLUDE_DIR=/nfs/LIBS/LIBS/CUDA_GDK/352.79/usr/include/nvidia/gdk
```

Con respecto a la variante anterior, la diferencia es el uso de los parámetros DNVML_LIBRARY y DNVML_INCLUDE_DIR que indican las rutas donde localizar las librerías de NVML.

El resto de pasos son los comunes al resto de instalaciones:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.7. Memoria distribuida con OpenMPI

Esta variante de la instalación hace uso de OpenMPI, que como ya se ha comentado, es una librería de código abierto que implementa el estándar MPI, la interfaz de paso de mensajes. Como en los casos anteriores, la instalación comienza por ejecutar *cmake* con los parámetros adecuados:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
OPENMPI/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
OPENMPI/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_GPU=OFF -
DGMX_MPI=ON -DMPI_CXX_COMPILER=/nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpicxx -
DMPI_C_COMPILER=/nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpicc
```

Donde, al igual que en el caso de MVAPICH2, el parámetro DGMX_MPI=ON indica a *cmake* que compile Gromacs con soporte a memoria distribuida, y los parámetros DMPI_CXX_COMPILER y DMPI_C_COMPILER determinan dónde se deben localizar los nuevos compiladores para OpenMPI.

Se prosigue pues de la misma forma que en los casos anteriores:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.8. Memoria distribuida con OpenMPI y GPU

En este caso se compilará Gromacs con soporte para utilizar las GPUs de las que dispongan los nodos implicados en la ejecución. Se comienza por cmake:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
OPENMPI-GPU/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-distribuida-OPENMPI-
GPU/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_MPI=ON -
DMPI_CXX_COMPILER=/nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpicxx -
```

```
DMPI_C_COMPILER=/nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpicc -DGMX_GPU=ON -
DCUDA_TOOLKIT_ROOT_DIR=/nfs/LIBS/LIBS/CUDA/CURRENT/
```

Y se prosigue con el resto de instrucciones que dan por concluida la instalación:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.9. Memoria distribuida con OpenMPI, GPU y NVML

La última instalación se corresponde con Gromacs con soporte para GPU y NVML, por lo que se procede con la herramienta *cmake*:

```
$ cmake .. -DREGRESSIONTEST_PATH=/nfs/alumnos/andiaro/gromacs-mem-distribuida-
OPENMPI-GPU-NVML/source-code/regressiontests-5.1.2/ -
DCMAKE_INSTALL_PREFIX=/nfs/alumnos/andiaro/gromacs-mem-distribuida-OPENMPI-
GPU-NVML/gromacs-installed/ -DGMX_FFT_LIBRARY=fftw3 -
DCMAKE_PREFIX_PATH=/nfs/LIBS/LIBS/FFTW/3.3.3/SINGLE/ -DGMX_MPI=ON -
DMPI_CXX_COMPILER=/nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpicxx -
DMPI_C_COMPILER=/nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpicc -DGMX_GPU=ON -
DCUDA_TOOLKIT_ROOT_DIR=/nfs/LIBS/LIBS/CUDA/CURRENT/ -
DNVML_LIBRARY=/usr/lib64/libnvidia-ml.so -
DNVML_INCLUDE_DIR=/nfs/LIBS/LIBS/CUDA_GDK/352.79/usr/include/nvidia/gdk
```

Y se concluye la instalación con:

```
$ make -j $(cat /proc/cpuinfo | grep -i processor | wc -l)

$ make check

$ make install
```

## 3.10. Conclusiones acerca de la instalación de Gromacs

Una vez concluido el proceso de instalación de Gromacs en las variantes mencionadas arriba, se ha podido observar que la dificultad de la instalación radica en la

obtención de las librerías a utilizar, y posteriormente en indicar a *cmake* la ruta de estas librerías, puesto que para personas menos experimentadas con la terminal, puede resultar tedioso y confuso. No obstante, la documentación de Gromacs ayuda sobremanera en este apartado, ya que cada uno de los pasos está documentado de forma clara y concisa.

Así pues, dejando al margen el aspecto anterior, la instalación de Gromacs no resulta distinta de la de cualquier otro software compilado a partir de su código fuente.

# 4. Ejecución de Gromacs

El objetivo de este capítulo es presentar cómo llevar a cabo la ejecución de Gromacs en las diferentes variantes compiladas en el capítulo anterior. Al iniciar la ejecución, Gromacs posee la capacidad de detectar los tipos de aceleración a los que se les ha dado soporte durante la compilación del programa, por lo que eso facilita enormemente al usuario la tarea de ejecutar la aplicación.

Debido a ello, la ejecución de Gromacs para el caso de memoria compartida será idéntica a la ejecución de memoria compartida utilizando aceleración por GPU, e idéntica a su vez a la ejecución de memoria compartida con aceleración por GPU y NVML. Por ello, únicamente se describirá en este capítulo la ejecución para uno de los tres casos de memoria compartida.

Con respecto a memoria distribuida ocurre lo mismo, es decir, la ejecución de Gromacs sin aceleración por GPU es idéntica a la ejecución con ella, e idéntica a su vez a la ejecución con GPU y NVML, por lo que la ejecución descrita más adelante en este capítulo será válida para cualquiera de los tres casos.

Para la ejecución de Gromacs en cada una de sus variantes se hará uso de un archivo de test llamado *ion_channel*, que consiste en un sistema molecular de tamaño intermedio de aproximadamente 140.000 átomos y que, por lo tanto, representa una carga promedio de la que una ejecución real de Gromacs podría producir. A todo esto, y con la intención de poder manejar cifras de tiempo no excesivamente grandes en los resultados, se ha considerado la ejecución de Gromacs con un argumento de entrada de exactamente 200.000 pasos. Para llegar a esta cifra se han realizado ciertas pruebas que han consistido en ejecutar Gromacs con dos nodos implicados, con 6 procesos MPI por cada nodo y 4 threads OpenMP por cada proceso. Además, se ha hecho uso de CUDA y NVML. El resultado ha sido el representado por la siguiente gráfica, donde se observa que con 200.000 pasos, el tiempo de ejecución es de poco más de 1600 segundos, una cifra que se maneja cómodamente en un análisis de prestaciones.

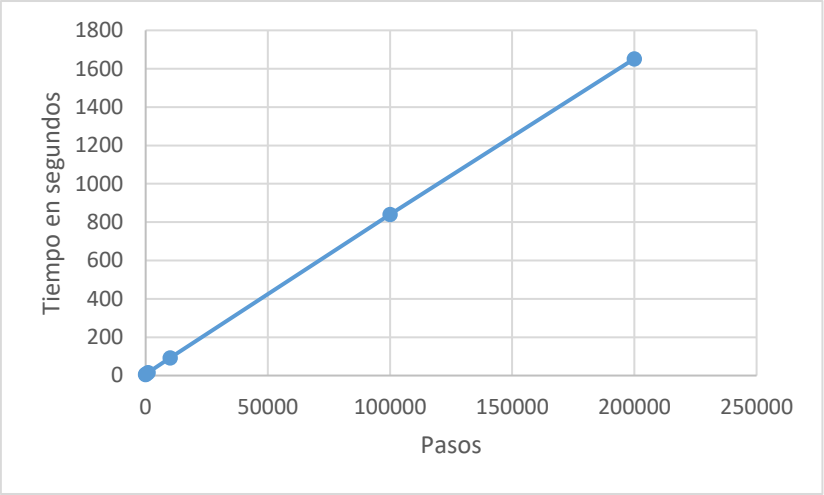


*Gráfico 1: evolución del tiempo de ejecución en función de los pasos ejecutados*

## 4.1. Memoria compartida

La ejecución de Gromacs en memoria compartida es relativamente sencilla. La idea es lanzar el ejecutable generado durante la instalación e indicarle a su vez el número de threads MPI y el número de threads OpenMP por cada thread MPI, el fichero de test que se desea ejecutar, el número de pasos que se desean realizar y si se desea asignar de forma permanente los hilos a los cores. Así pues, la ejecución quedaría como sigue:

```
$ ~/gromacs-mem-compartida/gromacs-installed/bin/gmx mdrun -ntmpi 2 -ntomp 6 -
s ~/testGromacs/ion_channel.tpr -nsteps 200000 -pin on
```

Donde:

- *gmx*: binario principal de Gromacs.
- *mdrun*: programa que lee el fichero indicado con el parámetro -s y distribuye la carga entre los threads finalmente lanzandos.
- *-ntmpi*: número de threads MPI. Si no se especifica, se lanza un thread MPI por cada GPU, y si no hay disponible ninguna, se lanza un thread MPI por cada core.
- *-ntomp*: número de threads OpenMP por cada thread MPI. Si no se especifica, se lanza un thread OpenMP por cada core disponible. No obstante, este parámetro también puede ser especificado con la variable de entorno OMP_NUM_THREADS.
- *-nsteps*: número de pasos a ejecutar.

- *-pin on*: asignar de forma permanente cada hilo a un núcleo físico (opción necesaria únicamente si no se lanzan tantos hilos como núcleos tiene la máquina). De esta forma se impide que los hilos migren a otros núcleos durante la ejecución, evitando así una posible pérdida de rendimiento.

Cuando se ejecuta Gromacs, en los resultados arrojados por la terminal se proporciona cierta información relativa al rendimiento en caso de que existan previsiones de que éste puede no ser óptimo. De esta forma, al lanzar Gromacs con más de 6 threads OpenMP por cada Thread-MPI, se proporciona un aviso en el que se especifica que el rendimiento obtenido con esa configuración podría no ser óptimo, y que la configuración óptima suele estar entre 2 y 6 threads OpenMP por cada thread MPI.

## 4.2. Memoria distribuida con MVAPICH2

En este caso, el binario a ejecutar ya no será el ejecutable de Gromacs propiamente dicho, si no *mpirun_rsh*, que constituye el binario principal de la librería MVAPICH2. Así pues, este ejecutable será el que finalmente lance la ejecución del binario de Gromacs pero repartiendo la carga entre los nodos implicados en la ejecución:

```
$ /nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpirun_rsh -ssh -np 16 -hostfile nodos
MV2_ENABLE_AFFINITY=0 MV2_SMP_USE_LIMIC2=1 MV2_IBA_HCA=mlx4_0 MV2_NUM_PORTS=1
~/gromacs-mem-distribuida-MVAPICH2/gromacs-installed/bin/gmx_mpi mdrun -s
~/testGromacs/ion_channel.tpr -nsteps 2000 -ntomp 3 -pin on
```

Por lo tanto, los parámetros usados durante la ejecución son los siguientes:

- *–ssh*: indica que la conexión entre los nodos se realiza mediante el protocolo ssh.
- *–np 16*: indica que la ejecución se llevará a cabo mediante los nodos del clúster indicados en el fichero *nodos* (np indica el número de procesos y esos procesos serán repartidos entre los nodos que se le asignen) pero creando entre todos los nodos 16 procesos MPI, donde cada uno gestionará el número de hilos OpenMP especificados en el parámetro *-ntomp*, con objeto de minimizar el coste de la gestión de un número elevado de hilos.
- *-hostfile*: fichero de texto en el que se indica, línea por línea, la lista de nodos que intervendrán en la ejecución. Nótese que en caso de querer

ejecutar más procesos que nodos, el nombre de cada nodo deberá repetirse tantas veces como procesos se desee asignar a ese nodo. Este parámetro es opcional, puesto que es posible lanzar Gromacs indicando la lista de nodos involucrados separados por espacios en blanco a continuación del parámetro *np*, pero usar un fichero de texto con el nombre de los nodos resulta más versátil.

- *MV2_ENABLE_AFFINITY=0*: determina que MVAPICH2 ejecute cada hilo de ejecución sobre un núcleo del procesador por separado. En caso de no usar este parámetro, MVAPICH2 asignaría todos los hilos del mismo proceso a un solo núcleo del procesador.
- *MV2_SMP_USE_LIMIC2=1*: indica la intención de usar la librería de alto rendimiento LIMIC2 en las comunicaciones intra-nodo.
- *MV2_IBA_HCA=mlx4_0*: indica que se desea usar la variante de InfiniBand FDR.
- *MV2_NUM_PORTS=1*: determina la intención de usar un solo puerto de red de las tarjetas InfiniBand.

## 4.3. Memoria distribuida con OpenMPI

En este último caso, se pretende ejecutar Gromacs entre varios nodos usando la librería OpenMPI. Así pues, su ejecución resultaría como sigue:

```
$ /nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpirun -np 16 --hostfile nodos ~/gromacs-mem-distribuida-OPENMPI/gromacs-installed/bin/gmx_mpi mdrun -ntomp 3 -s ~/testGromacs/ion_channel.tpr -nsteps 200000 -pin on
```

Donde destacan las siguientes opciones:

- *mpirun*: binario de OpenMPI.
- *-np 16*: indica, al igual que en el caso anterior, el número de procesos MPI que se lanzarán en la ejecución. Cada proceso MPI será el encargado de gestionar tantos threads OpenMP como los especificados con el parámetro *ntomp*.
- *--hostfile*: indica el nombre del fichero en el que aparecen, línea por línea, los nombres de los nodos implicados en la ejecución. Al igual que en el caso de MVAPICH2, es posible obviar este parámetro ejecutando

OpenMPI con el parámetro *-H*, indicando a continuación la lista de nodos involucrados pero separados por comas.

Para conocer la forma en que se ejecuta Gromacs en todas las variantes instaladas en este trabajo, se ha hecho uso, una vez más, de la documentación oficial de Gromacs y de la documentación asociada a MVAPICH2 [9] y OpenMPI [8].

Como puede observarse al leer esta sección, la ejecución de Gromacs en memoria compartida es extremadamente sencilla. En esencia, se necesita acceder al binario de Gromacs y especificar ciertos parámetros básicos que permiten personalizar la ejecución.

En lo que respecta a ejecutar Gromacs en memoria distribuida, ya no es tanto el proceso de ejecución de Gromacs el que determina su dificultad, si no el proceso de ejecución de la librería MPI que se haya usado, puesto que será el binario de esta librería el que ejecutará finalmente el binario de Gromacs.

# 5. Pruebas de rendimiento

En este capítulo se pretende realizar una presentación de las métricas obtenidas en la monitorización de la ejecución de Gromacs en todas sus variantes. Además de la métrica clásica del tiempo de ejecución final, Gromacs posee una métrica propia de rendimiento basada en los nanosegundos simulados por cada día de ejecución del programa, y que resultará por tanto en una variable que indicará de forma clara el rendimiento final obtenido.

Además, se medirá la potencia de cada nodo y de cada GPU en el caso de utilizarla. Por lo tanto, el primer paso es generar un fichero de texto en el que se introducirán línea por línea tantos nodos como se quieran monitorizar. Una vez hecho eso, se copia ese fichero al directorio /nfs/energy del clúster, de esta forma se activará la medición de energía. Una vez terminada la monitorización, únicamente será necesario eliminar el fichero de texto. Con esto, se generará un directorio en el que aparecerá tantos sub-directorios como nodos se han monitorizado. En cada subdirectorio quedarán recogidos los datos de consumo tanto del nodo como de la GPU.

## 5.1. Memoria compartida

Las pruebas de memoria compartida realizadas en este trabajo han consistido en la ejecución de Gromacs en las variantes de memoria compartida sin GPU, memoria compartida con GPU, memoria compartida usando GPU y la librería NVML, y por último, memoria compartida usando múltiples GPUs. Los resultados obtenidos son los que se muestran a continuación:

### 5.1.1. Memoria compartida sin GPU

La ejecución de Gromacs en memoria compartida ha obtenido las siguientes métricas de rendimiento y potencia en vatios del nodo en función del número de threads OpenMP lanzados, recordando que el nodo utilizado dispone de 12 núcleos físicos y 2 núcleos lógicos por cada núcleo físico.

| Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo |
|---|---|---|---|
| **1** | 62896 | 0,687 | 154,57 |
| **2** | 32160 | 1,343 | 175,12 |
| **4** | 16560 | 2,607 | 185,91 |
| **8** | 8678 | 4,979 | 224,24 |
| **12** | 6395 | 6,756 | 242,43 |
| **24** | 5591 | 7,729 | 263,62 |

*Tabla 1: métricas en memoria compartida*

Se adjuntan a continuación los gráficos correspondientes a la tabla 1, donde se aprecia un claro aumento del rendimiento a medida que aumenta el número de threads utilizados. No obstante, puede apreciarse también una desaceleración en el aumento del rendimiento a medida que aumenta el número de hilos empleados, y esto es debido a la sobrecarga en las comunicaciones intra-nodo a medida que aumentan los hilos en ejecución. En el gráfico 3 puede verse el incremento de potencia conforme aumentan los núcleos usados del nodo.

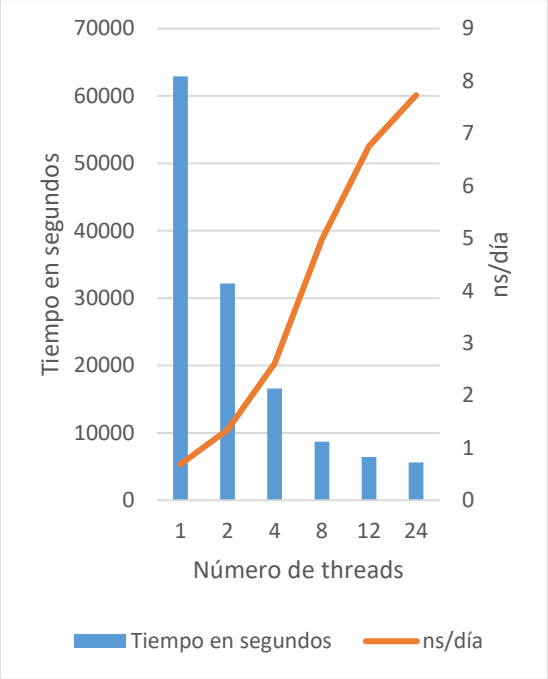


*Gráfico 2: tiempo en segundos en memoria compartida*

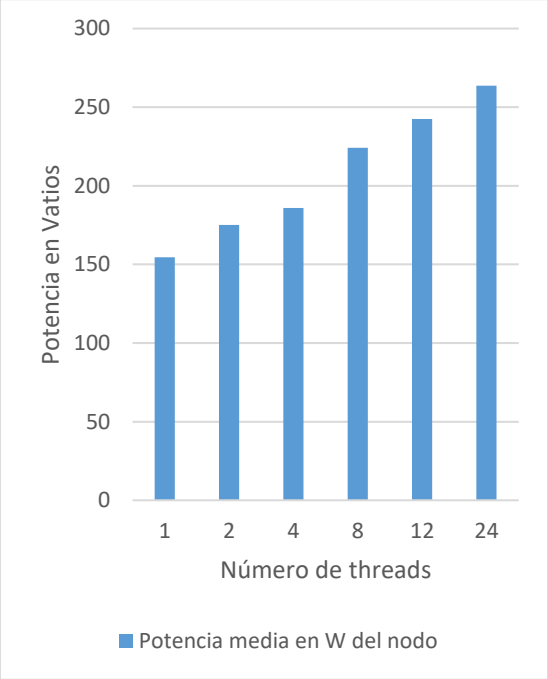


*Gráfico 3: potencia en Vatios en memoria compartida*

### 5.1.2. Memoria compartida con GPU

En este caso, se ejecuta la variante de Gromacs compilada para hacer uso de la tarjeta gráfica conectada al nodo mediante la librería CUDA. Se han obtenido los siguientes datos de rendimiento y potencia ejecutando esta variante:

| Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo | Potencia media en W de la GPU |
|---|---|---|---|---|
| **1** | 14000 | 3,086 | 217,55 | 100,60 |
| **2** | 8190 | 5,276 | 255,69 | 92,38 |
| **4** | 4466 | 9,676 | 263,82 | 96,79 |
| **8** | 2707 | 15,966 | 301,49 | 99,57 |
| **12** | 2367 | 18,261 | 306,62 | 101,01 |
| **24** | 2231 | 19,383 | 323,25 | 94,01 |

*Tabla 2: métricas en memoria compartida con GPU*

A continuación, se muestran las representaciones gráficas asociadas a la tabla 2:

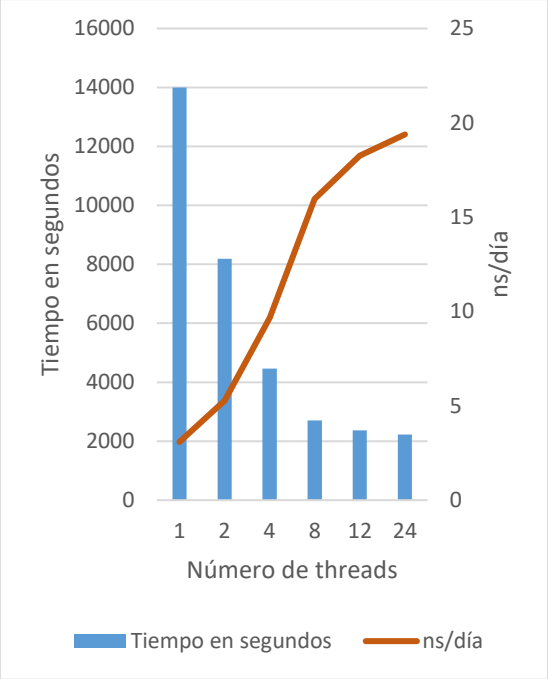


*Gráfico 4: tiempo en segundos en memoria compartida con GPU*

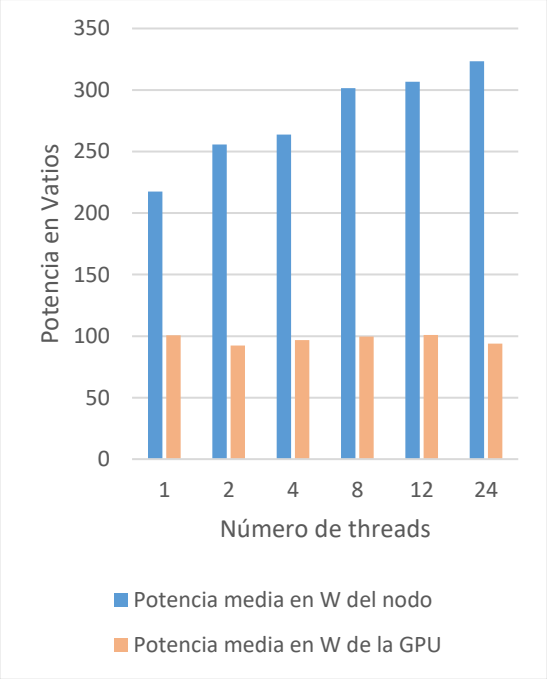


*Gráfico 5: ns/día en memoria compartida con GPU*

Al igual que ocurre con la variante sin GPU, se aprecia una clara mejora del rendimiento cuando se incrementa el número de threads, con la diferencia de que en este caso los threads gestionan los procesos lanzados llevándolos a la GPU con objeto de mejorar el rendimiento. En lo relativo a la potencia, ésta se incrementa en la CPU conforme aumenta el número de hilos, mientras que la potencia de la GPU se mantiene relativamente constante, puesto que la cantidad de trabajo asignado a la misma es similar en todos los casos.

### 5.1.3. Memoria compartida con GPU y NVML

En esta variante de Gromacs se usa la librería NVML para obtener una ligera mejora en el rendimiento. Los datos obtenidos son los siguientes (se acompañan gráficos que ilustran estos datos):

| Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo | Potencia media en W de la GPU |
|---|---|---|---|---|
| **1** | 13994 | 3,087 | 224,97 | 108,12 |
| **2** | 8080 | 5,348 | 261,55 | 114,67 |
| **4** | 4441 | 9,730 | 289,31 | 108,64 |
| **8** | 2705 | 15,971 | 314,70 | 111,62 |
| **12** | 2223 | 19,451 | 332,25 | 114,13 |
| **24** | 2177 | 19,860 | 344,97 | 114,91 |

*Tabla 3: métricas en memoria compartida con GPU y NVML*

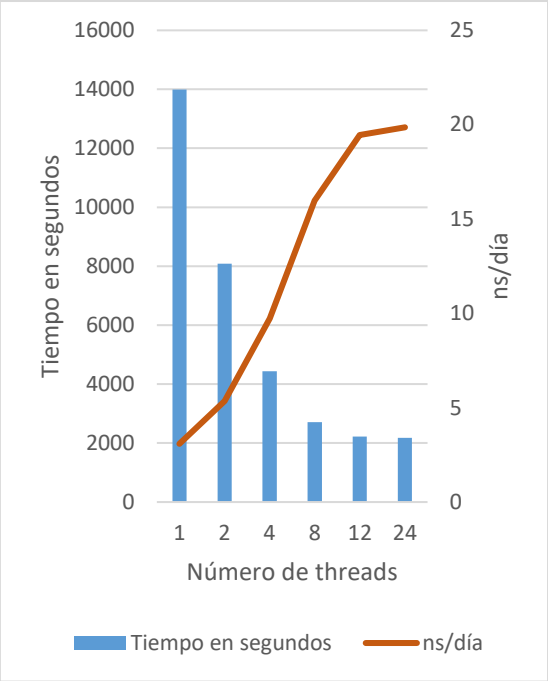


*Gráfico 6: tiempo en segundos en memoria compartida con GPU y NVML*

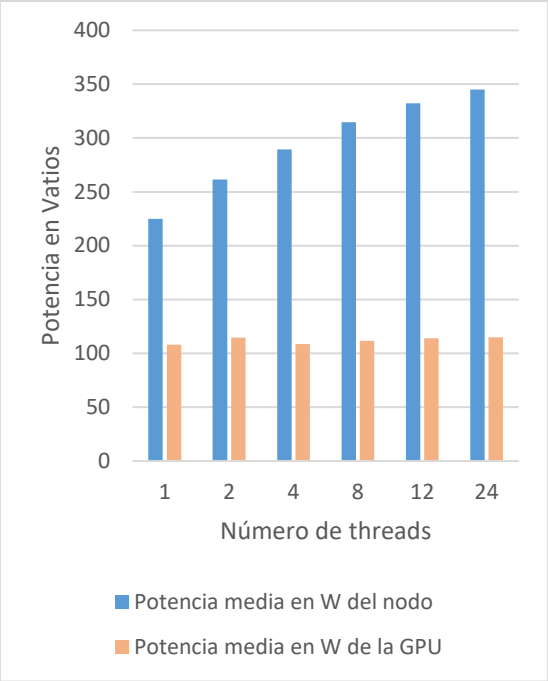


*Gráfico 7: ns/día en memoria compartida con GPU y NVML*

Como en los casos anteriores, a mayor número de threads mayor rendimiento, apreciándose sin embargo una desaceleración en el incremento del rendimiento en el paso de los 12 a los 24 threads.

### 5.1.4. Memoria compartida con múltiples GPUs

En esta prueba se pretende ejecutar Gromacs en memoria compartida de forma que sean utilizadas un número creciente de GPUs. La ejecución se ha llevado a cabo con un número creciente de threads MPI y un número constante de threads OpenMP por thread MPI, de forma que en cada ejecución el número de procesos totales coincida con el número de GPUs en uso. De esta forma, en todas las ejecuciones, cada uno de los procesos se ejecuta de forma exclusiva en una GPU.

Los resultados son los de la tabla siguiente:

| Número de GPUs | Tiempo en segundos | ns/día |
|---|---|---|
| **1** | 5736 | 7,534 |
| **2** | 3414 | 12,664 |
| **3** | 2566 | 16,857 |
| **4** | 2081 | 20,709 |

*Tabla 4: rendimiento en múltiples GPUs en memoria compartida*

El gráfico asociado al rendimiento de esta prueba es el siguiente:

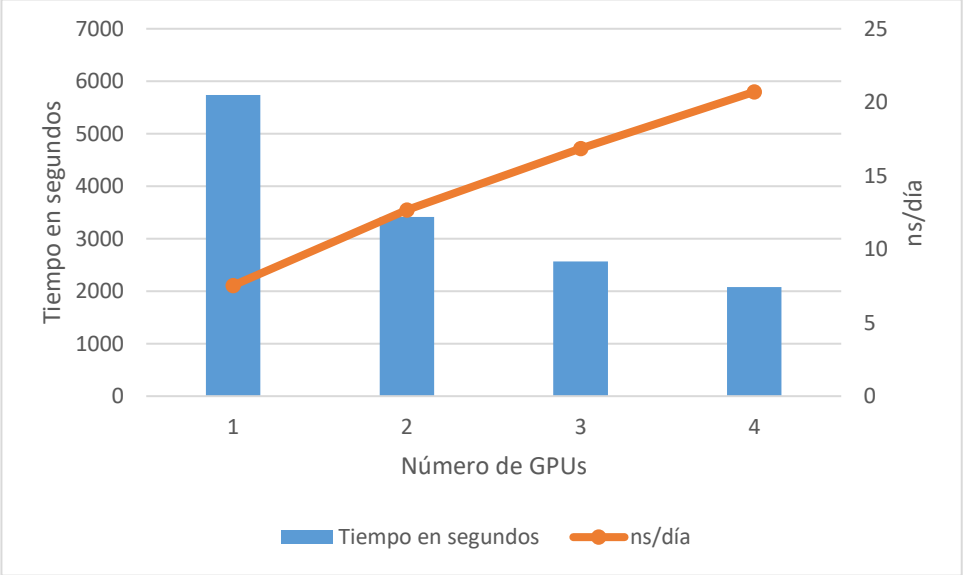


*Gráfico 8: tiempo en segundos y ns/día usando múltiples GPUs en memoria compartida*

Donde se aprecia un claro aumento de rendimiento conforme sube el número de GPUs usadas, pese a que también es evidente una desaceleración en el incremento del rendimiento.

### 5.1.5. Comparativa de memoria compartida

El siguiente gráfico muestra, para los tipos de memoria compartida sin GPU y con una única GPU, el tiempo de ejecución en segundos y el rendimiento en ns/día. Puede apreciarse que la diferencia de rendimiento entre la versión en memoria compartida con y sin GPU es significativamente elevada, mientras que la diferencia entre usar NVML o

no usarlo es mucho menor, pero aun así relevante si se desea obtener el máximo rendimiento.

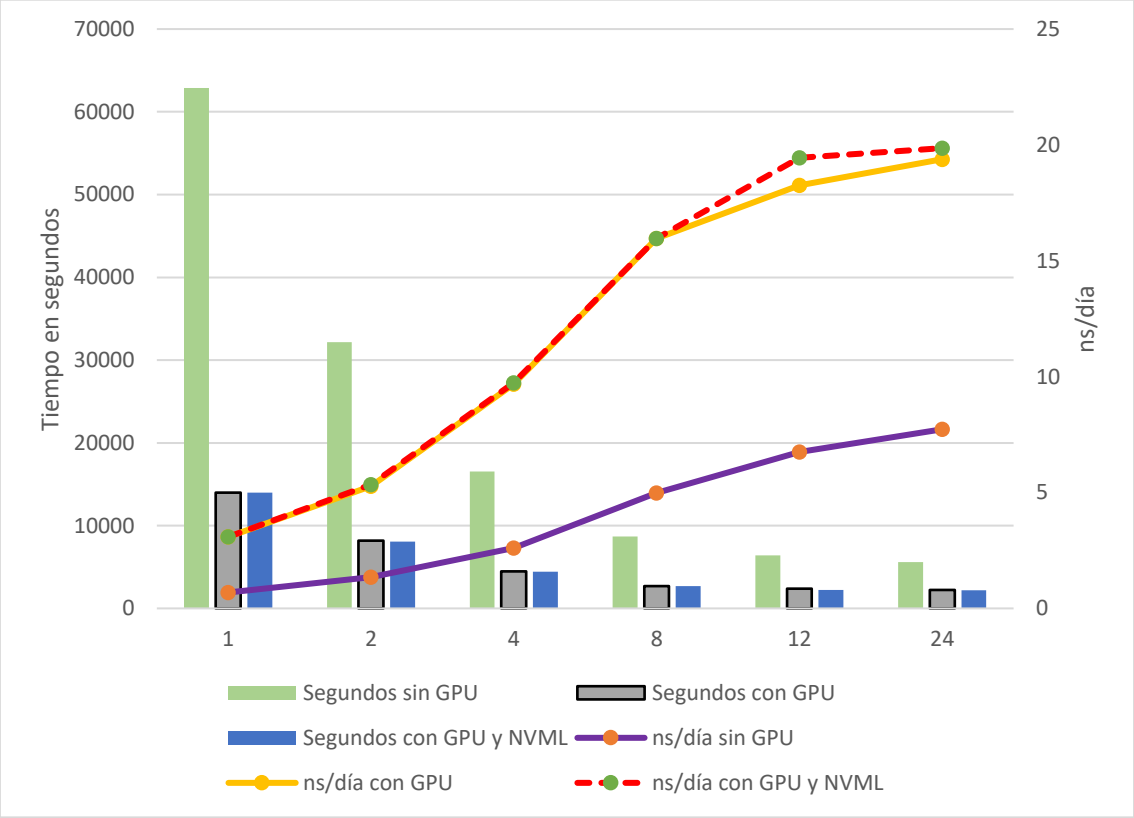


*Gráfico 9: comparativa entre los distintos tipos de memoria compartida*

A continuación, se procede a presentar la aceleración obtenida en las pruebas de rendimiento. La aceleración o speed-up en un programa paralelo es la ganancia en velocidad que se obtiene con respecto a la versión secuencial del mismo programa. En este caso, se considera la versión secuencial como la ejecución de Gromacs usando un único thread. La aceleración se determina mediante el cociente entre el tiempo empleado en la versión secuencial y el tiempo empleado en la versión paralela del programa:

$$S(n,p) = \frac{t(n)}{t(n,p)}$$

Así pues, se obtienen las siguientes aceleraciones en cada una de las variantes de Gromacs en memoria compartida:

| Número de threads | Memoria compartida | Memoria compartida con GPU | Memoria compartida con GPU y NVML |
|---|---|---|---|
| **1** | 1,00 | 1,00 | 1,00 |
| **2** | 1,96 | 1,71 | 1,73 |
| **4** | 3,80 | 3,13 | 3,15 |
| **8** | 7,25 | 5,17 | 5,17 |
| **12** | 9,84 | 5,91 | 6,30 |
| **24** | 11,25 | 6,28 | 6,43 |

*Tabla 5: relación de aceleraciones*

Y seguidamente se presenta la relación de eficiencias obtenidas en el caso de memoria compartida. La eficiencia de un programa paralelo indica el grado de aprovechamiento que dicho programa hace de un computador con capacidad para ejecutar programas paralelos. La eficiencia se determina mediante el cociente entre la aceleración obtenida y el número de threads involucrados en la ejecución:

$$E(n,p) = \frac{S(n,p)}{p}$$

Así pues, la relación de eficiencias obtenidas es la siguiente:

| Número de threads | Memoria compartida | Memoria compartida con GPU | Memoria compartida con GPU y NVML |
|---|---|---|---|
| **1** | 1,00 | 1,00 | 1,00 |
| **2** | 0,98 | 0,85 | 0,87 |
| **4** | 0,95 | 0,78 | 0,79 |
| **8** | 0,91 | 0,65 | 0,65 |
| **12** | 0,82 | 0,49 | 0,52 |
| **24** | 0,47 | 0,26 | 0,27 |

*Tabla 6: relación de eficiencias*

Por último, se presentan los gráficos asociados a las dos tablas anteriores, en la que se presentan los datos sobre aceleración y eficiencia:

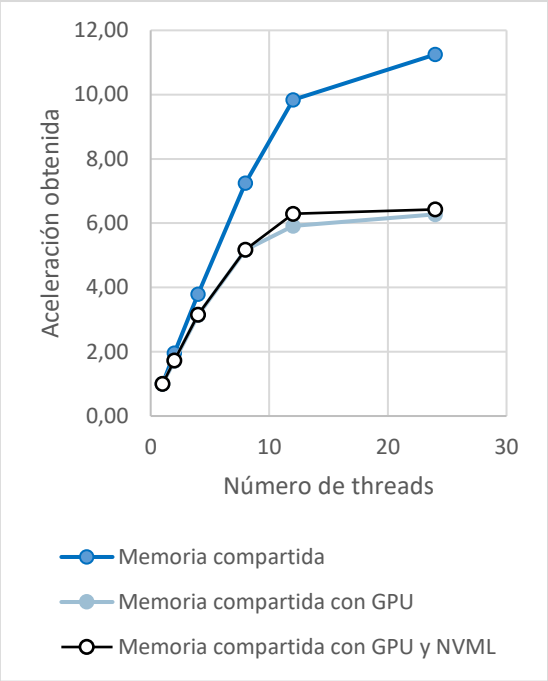


*Gráfico 10: aceleración en memoria compartida*

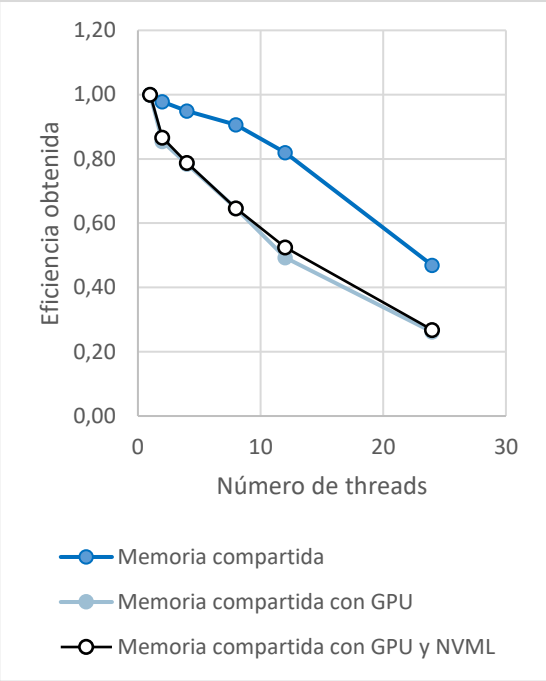


*Gráfico 11: eficiencia en memoria compartida*

En el gráfico 9 se observa la tendencia de la aceleración de cada una de las variantes de memoria compartida. En el caso de no usar GPU, la aceleración del procesador es mucho más pronunciada, pues no interviene ningún acelerador externo que le reste trabajo a los núcleos del procesador principal. Sin embargo, al usar GPU, ésta se lleva buena parte del trabajo a realizar, y por lo tanto el incremento de threads en el procesador principal afecta en menor medida.

Destaca también la baja aceleración obtenida cuando se ejecuta Gromacs usando 24 threads en memoria compartida. Esto es debido a que entra en juego la tecnología HyperThreading, que simula la existencia de dos núcleos lógicos por cada núcleo físico, aunque es evidente que la aceleración obtenida dista con mucho de la obtenida usando únicamente 12 threads mapeados a los 12 núcleos físicos.

En cuanto a la eficiencia, se aprecia claramente que el aprovechamiento del hardware decrece significativamente a medida que se incrementa el número de threads. Esto es debido al coste temporal que conlleva la gestión de un número mayor de hilos en ejecución.

A todo esto, se aprecia que la eficiencia obtenida con 12 threads es de un 82%, mientras que la obtenida con 24 threads es de un 47%, una clara diferencia que es debida al uso de la tecnología HyperThreading de Intel.

Por último, se presenta a continuación la relación de aceleraciones y eficiencias para el caso de memoria compartida usando múltiples GPUs y su gráfico asociado:

| Número de GPUs | Aceleración | Eficiencia |
|---|---|---|
| **1** | 1,00 | 1,00 |
| **2** | 1,68 | 0,84 |
| **3** | 2,24 | 0,75 |
| **4** | 2,76 | 0,69 |

*Tabla 7: aceleración y eficiencia con múltiples GPUs en memoria compartida*

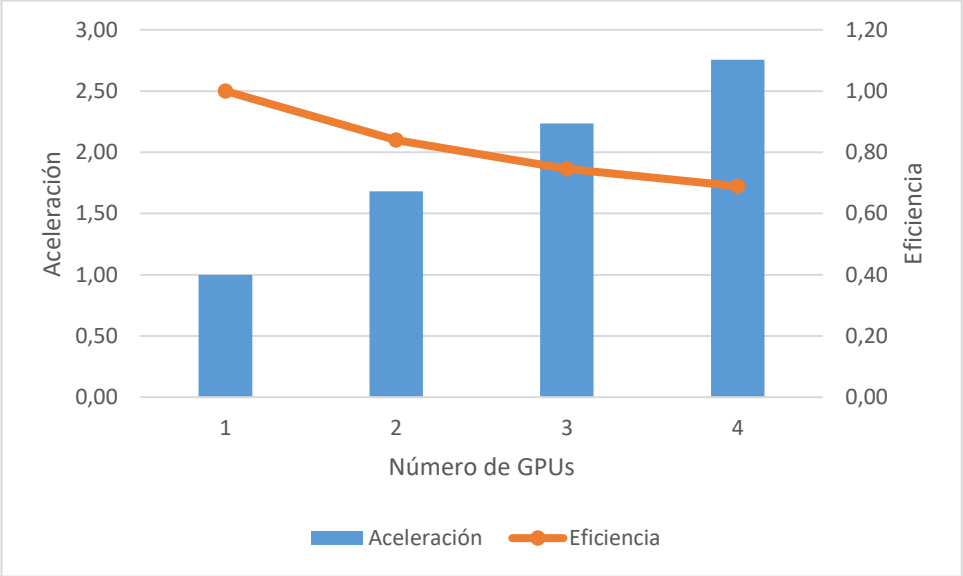


*Gráfico 12: aceleración y eficiencia usando múltiples GPUs en memoria compartida*

Donde se aprecia claramente que la aceleración se incrementa notablemente conforme aumenta el número de GPUs, pero muy por debajo de lo deseable. Esto es debido a un menor aprovechamiento del hardware, dato comprobable observando la línea decreciente de la eficiencia alcanzada.

## 5.2. Memoria distribuida

Para la realización de las pruebas en memoria distribuida se ha considerado efectuar en primer lugar un test de ancho de banda con objeto de comparar los resultados de las dos librerías MPI mencionadas en este trabajo, usando en ambos casos la red de alta velocidad InfiniBand FDR. Además, también se ha considerado efectuar en una de las dos librerías el mismo test pero usando la red Gigabit Ethernet para comprobar la diferencia de prestaciones con la red InfiniBand.

El test para OpenMPI se realizará entre los nodos mlxc3 y mlxc5 de la siguiente manera [12]:

```
$ /nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpirun --mca btl self,sm,openib --mca
btl_openib_if_include mlx4_0 -np 2 -H mlxc3,mlxc5 /nfs/alumnos/andiaro/OSU-
OpenMPI/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw
```

Y el test para MVAPICH2, también entre los mismos nodos, se ha efectuado como sigue [13]:

```
$ /nfs/LIBS/LIBS/MVAPICH2/2.0b/bin/mpirun_rsh -ssh -np 2 mlxc3 mlxc5
MV2_SMP_USE_LIMIC2=0 MV2_IBA_HCA=mlx4_0 MV2_NUM_PORTS=1
/nfs/alumnos/andiaro/OSU-MVAPICH2/libexec/osu-micro-
benchmarks/mpi/pt2pt/osu_bw
```

Por último, el test de velocidad para OpenMPI usando la red Gigabit Ethernet se ha ejecutado como sigue:

```
$ /nfs/LIBS/LIBS/OPENMPI/1.6.5/bin/mpirun --mca btl self,sm,tcp --mca
btl_tcp_if_include eth0 -np 2 -H mlxc3,mlxc5 /nfs/alumnos/andiaro/OSU-
OpenMPI/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw
```

Y los resultados de las tres pruebas se presentan en la siguiente tabla:

| Tamaño del mensaje en bytes | MVAPICH2 con InfiniBand FDR en MB/s | OpenMPI con InfiniBand FDR en MB/s | OpenMPI con Gigabit Ethernet en MB/s |
|---|---|---|---|
| 1 | 1,76 | 2,76 | 0,22 |
| 2 | 3,51 | 5,71 | 0,49 |
| 4 | 6,87 | 11,46 | 1,00 |
| 8 | 13,21 | 22,60 | 1,98 |
| 16 | 26,39 | 44,52 | 3,93 |
| 32 | 52,61 | 86,42 | 8,03 |
| 64 | 102,81 | 168,13 | 15,48 |
| 128 | 194,07 | 322,78 | 31,34 |
| 256 | 446,58 | 617,81 | 57,65 |
| 512 | 857,81 | 1198,98 | 79,73 |
| 1024 | 1585,68 | 2166,61 | 93,95 |
| 2048 | 2619,88 | 3740,60 | 100,24 |
| 4096 | 4315,19 | 5129,04 | 108,59 |
| 8192 | 5662,48 | 5533,39 | 111,59 |
| 16384 | 5705,08 | 5966,28 | 114,69 |
| 32768 | 5986,79 | 6185,50 | 116,42 |
| 65536 | 6150,47 | 6272,73 | 116,00 |
| 131072 | 6248,28 | 6312,46 | 116,76 |
| 262144 | 6305,08 | 6349,71 | 117,23 |
| 524288 | 6338,26 | 6362,18 | 117,49 |
| 1048576 | 6355,55 | 6370,97 | 117,59 |
| 2097152 | 6325,80 | 6330,59 | 117,64 |
| 4194304 | 6326,77 | 6336,20 | 117,66 |

*Tabla 8: resultados de la prueba de ancho de banda*

El gráfico asociado a estos datos se presenta a continuación, donde se observa un ligero aumento de la velocidad en el caso de OpenMPI con respecto a MVAPICH2 en todos los tamaños del mensaje. Sin embargo, y para el caso del uso de la red Ethernet, la velocidad queda muy por debajo de la alcanzada usando la red InfiniBand. Este resultado es el esperado dado el conocido ancho de banda máximo teórico de la red Gigabit Ethernet.

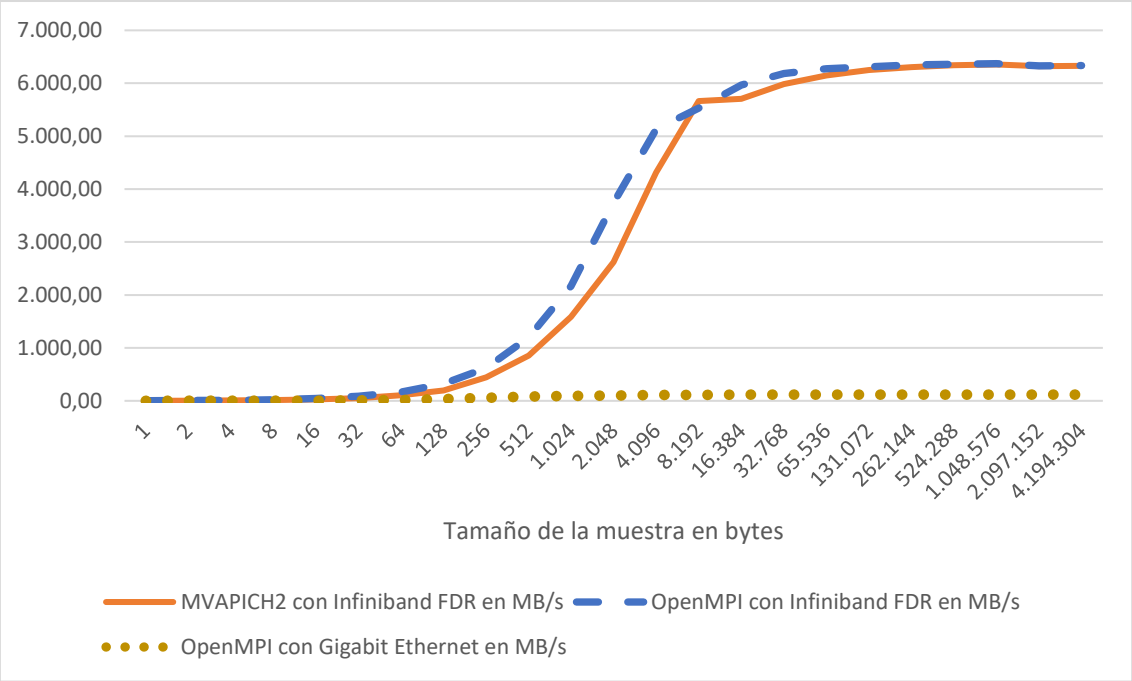


*Gráfico 13: test de ancho de banda entre MVAPICH2 con FDR, OpenMPI con FDR y OpenMPI con GbEth*

Por otra parte, se ha efectuado una prueba para comparar a OpenMPI con MVAPICH y ver la diferencia de rendimiento entre ambas. Para dicha prueba se han lanzado dos simulaciones del archivo de test *ion_channel* con 4 nodos y 4 procesos por nodo, sin GPU y 6 threads OpenMP por cada proceso MPI. El número de pasos efectuado ha sido de 100.000, y el resultado se muestra en la siguiente gráfica:

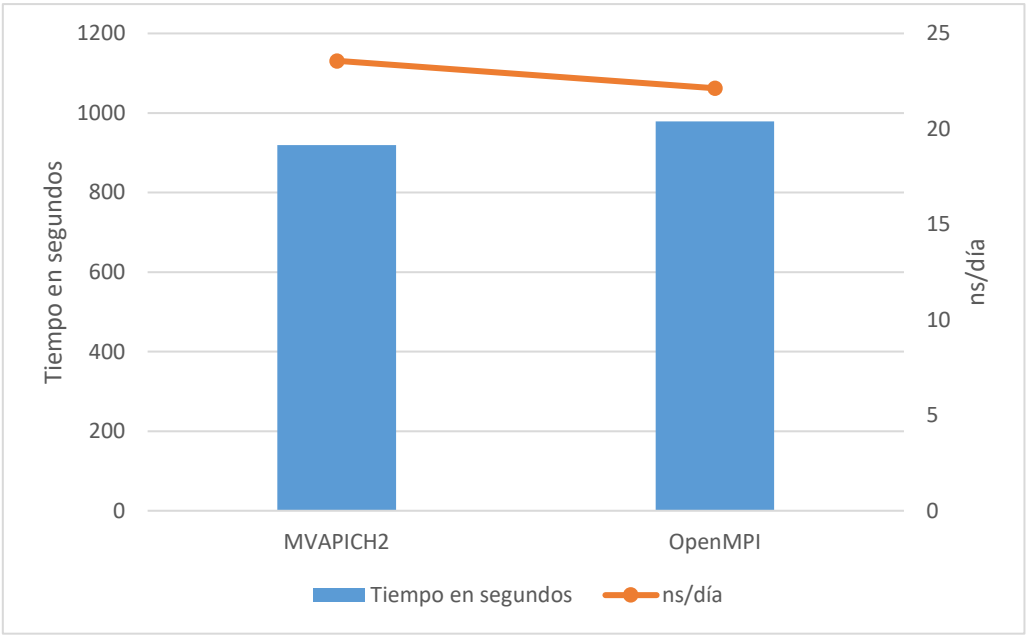


*Gráfico 14: tiempo en segundos y ns/día en una ejecución de Gromacs con MVAPICH2 y OpenMPI*

Donde, como se aprecia, MVAPICH2 supera en rendimiento a OpenMPI en pruebas reales. Esto es debido a que, en el caso de una prueba real, además de transferir datos rápidamente como ocurre en el test de velocidad, se debe gestionar el paso de los mensajes entre los nodos implicados.

Debido a que MVAPICH2 obtiene mejores resultados en pruebas reales, se ha optado por realizar el resto de pruebas con esta librería en este Trabajo Fin de Grado.

### 5.2.1. MVAPICH2 en un nodo

En esta prueba se ha realizado una ejecución de Gromacs con MVAPICH2 usando un único nodo, esto es, memoria distribuida en un único nodo, haciendo distinción entre usar únicamente CPU y usar además GPU. Los resultados son los que se muestran a continuación:

Usando CPU:

| Procesos | Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo |
|---|---|---|---|---|
| **2** | 6 | 6640 | 6,508 | 284,48 |
| **4** | 6 | 5801 | 7,449 | 227,98 |
| **12** | 1 | 6097 | 7,088 | 202,72 |

*Tabla 9: métricas con CPU*

Y usando GPU como acelerador:

| Procesos | Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo | Potencia media en W de la GPU |
|---|---|---|---|---|---|
| **2** | 6 | 2700 | 16,021 | 304,67 | 113,80 |
| **4** | 6 | 2716 | 15,923 | 306,45 | 98,41 |
| **12** | 1 | 2818 | 15,361 | 304,00 | 100,09 |

*Tabla 10: métricas con GPU*

A continuación, se presentan los gráficos asociados a los datos obtenidos en esta prueba:

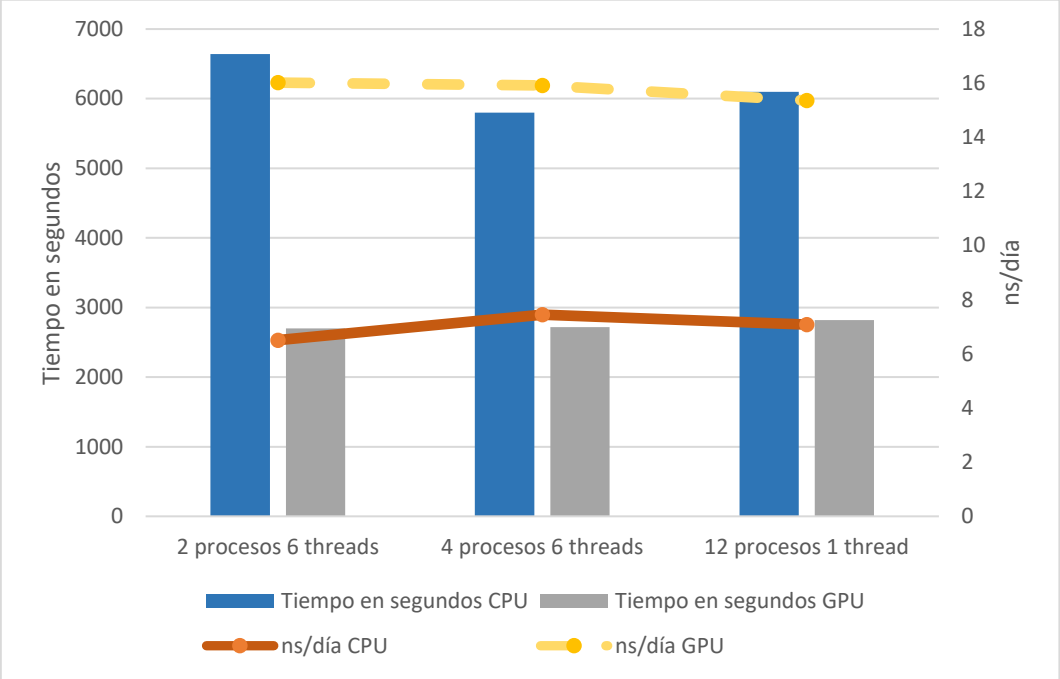


*Gráfico 15: rendimiento de MVAPICH2 en un nodo*

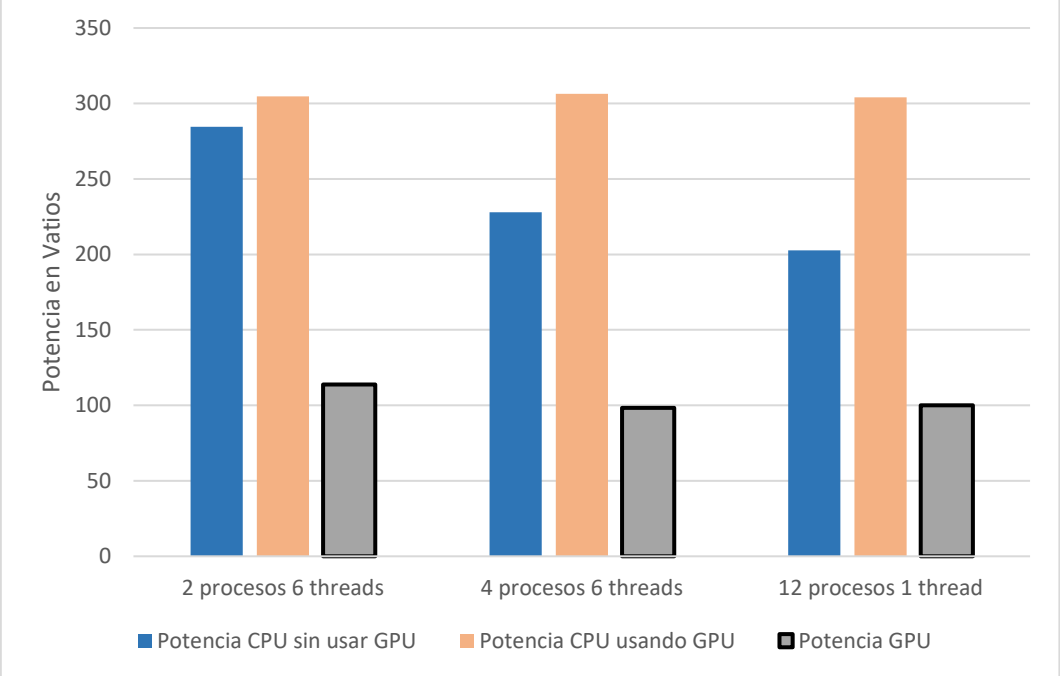


*Gráfico 16: potencia de MVAPICH2 en un nodo*

Como se aprecia en el gráfico del rendimiento, para todos los casos estudiados la versión más rápida es la que utiliza GPU. Por otro lado, se aprecian diferencias entre las

versiones ejecutadas variando el número de procesos MPI y threads OpenMP. De las tres versiones estudiadas, la que mayor rendimiento obtiene es la de 2 procesos y 6 threads, si bien en la versión GPU esta diferencia de rendimiento es menor.

### 5.2.2. MVAPICH2 en múltiples nodos

En esta prueba se ha realizado una ejecución de Gromacs con MVAPICH2 usando un número variable de nodos. Además, para cada caso se han lanzado distintas combinaciones de procesos y threads. La idea que subyace detrás de estas combinaciones es analizar el rendimiento en los casos en los que hay un solo thread por cada nodo, 12 threads por cada nodo ocupando los 12 cores físicos, y 24 threads por cada nodo haciendo uso de la tecnología HyperThreading, esto es, ocupando los 24 cores lógicos. Los resultados son los que se muestran a continuación:

Usando CPU:

| Nº Nodos | Procesos por nodo | Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo |
|---|---|---|---|---|---|
| **2** | 1 | 1 | 31954 | 1,352 | 149,32 |
| **2** | 2 | 6 | 3599 | 12,180 | 272,66 |
| **2** | 4 | 6 | 3314 | 13,046 | 291,38 |
| **4** | 1 | 1 | 15942 | 2,710 | 147,84 |
| **4** | 2 | 6 | 1993 | 21,706 | 281,52 |
| **4** | 4 | 6 | 1778 | 24,335 | 298,24 |
| **8** | 1 | 1 | 8121 | 5,321 | 146,91 |
| **8** | 2 | 6 | 1153 | 37,549 | 218,79 |
| **12** | 1 | 1 | 5228 | 7,680 | 153,44 |

*Tabla 11: métricas en MVAPICH usando CPU*

Y usando GPU:

| Nº Nodos | Procesos por nodo | Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo | Potencia media en W de la GPU |
|---|---|---|---|---|---|---|
| **2** | 1 | 1 | 7402 | 5,839 | 214,85 | 79,21 |
| **2** | 2 | 6 | 1645 | 26,316 | 240,06 | 97,02 |
| **2** | 4 | 6 | 3318 | 13,032 | 307,96 | 102,32 |
| **4** | 1 | 1 | 4420 | 9,780 | 217,40 | 77,12 |
| **4** | 2 | 6 | 937 | 46,313 | 243,78 | 79,84 |
| **4** | 4 | 6 | 1013 | 42,814 | 301,43 | 85,92 |
| **8** | 1 | 1 | 2587 | 16,723 | 224,62 | 78,52 |
| **8** | 2 | 6 | 628 | 69,116 | 302,28 | 81,32 |
| **12** | 1 | 1 | 1841 | 23,535 | 232,42 | 69,48 |

*Tabla 12: métricas en MVAPICH usando GPU*

La representación gráfica de estos resultados es la que se muestra a continuación:

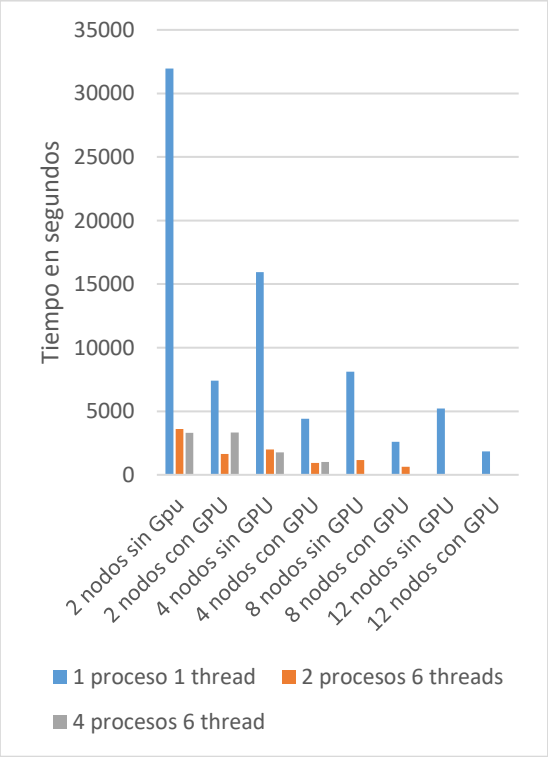


*Gráfico 17: tiempo en segundos en MVAPICH2 con diferentes nodos*

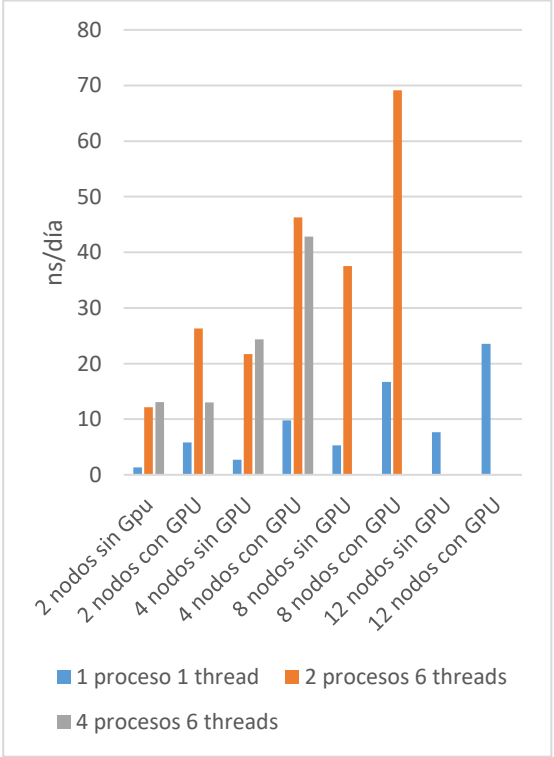


*Gráfico 18: ns/día en MVAPICH2 con diferentes nodos*

Al observar estos gráficos, se aprecia una clara línea descendiente en el tiempo conforme va aumentando el número de nodos. Además, como se viene observando en las pruebas anteriores, el rendimiento con GPU es mucho mayor que sin GPU. Por otra parte, en las variantes de múltiples procesos y threads, se observa que en las configuraciones sin GPU, la opción más óptima es la de 4 procesos y 6 threads, mientras que para las configuraciones con GPU, la mejor opción es la de 2 procesos y 6 threads.

A todo esto, se aprecia que el uso de la tecnología HyperThreading resulta en una ligera mejora del rendimiento cuando no interviene una GPU en la ejecución, mientras que en el caso contrario la tecnología HyperThreading resulta en una reducción de prestaciones, además de producir un incremento de la energía consumida, tal y como puede apreciarse en el gráfico siguiente:

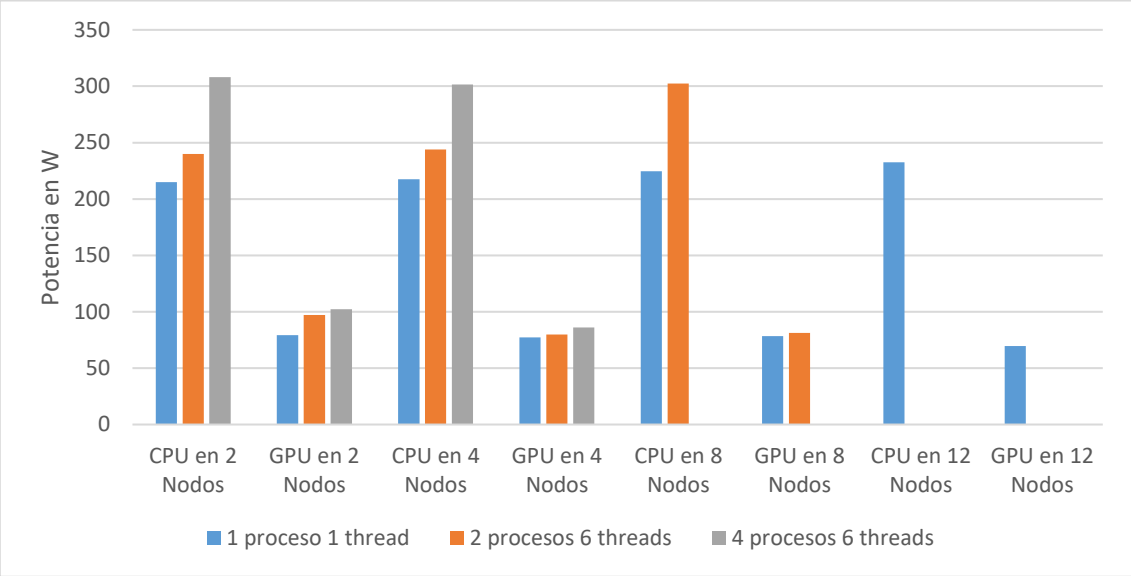


*Gráfico 19: potencia en W en MVAPICH2 con diferentes nodos*

Nótese que en los casos en los que se superan los 24 procesos MPI totales no hay resultado disponible. Esto es debido a que para esta prueba en particular, Gromacs alcanza el límite de escalabilidad dado que es un sistema 24 particle-particle, y por lo tanto el número de procesos final que ejecuta no es el esperado y los resultados no son útiles para el análisis de rendimiento.

### 5.2.3. MVAPICH2 con múltiples GPUs

En esta prueba se pretende realizar una ejecución de Gromacs en un solo nodo que dispone de 4 GPUs. Por lo tanto, la idea es comprobar cómo afecta el uso de un número variable de GPUs a los resultados de rendimiento y potencia en memoria distribuida.

Los datos obtenidos en la prueba se presentan en la siguiente tabla, y se acompañan con sus correspondientes gráficos.

| Nº GPUs | Procesos | Número de threads OpenMP | Tiempo en segundos | ns/día | Potencia media en W del nodo | Potencia media en W de la GPU |
|---|---|---|---|---|---|---|
| **2** | 1 | 1 | 12992 | 3,326 | 316,03 | 129,89 |
| **2** | 2 | 6 | 2237 | 19,328 | 345,32 | 106,24 |
| **2** | 4 | 6 | 2216 | 19,520 | 380,13 | 95,42 |
| **4** | 1 | 1 | 13031 | 3,316 | 323,42 | 117,01 |
| **4** | 2 | 6 | 2225 | 19,442 | 443,42 | 111,95 |
| **4** | 4 | 6 | 1967 | 22,007 | 644,29 | 102,51 |

*Tabla 13: métricas de MVAPICH2 usando múltiples GPUs*

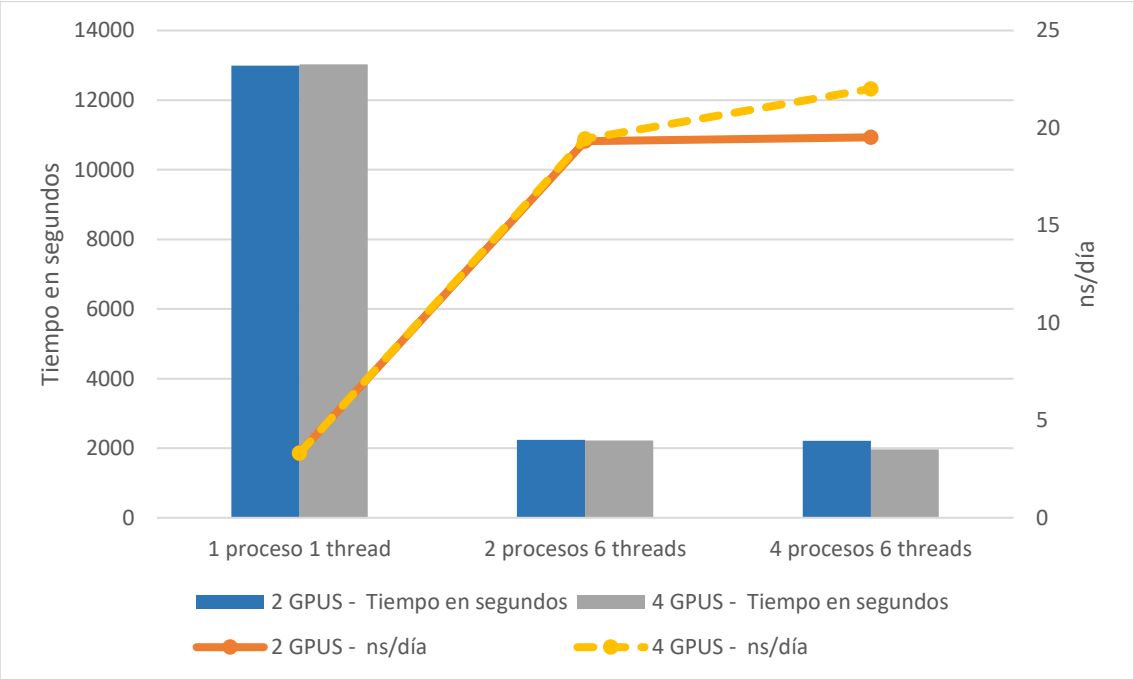


*Gráfico 20: rendimiento de MVAPICH2 con múltiples GPUs*

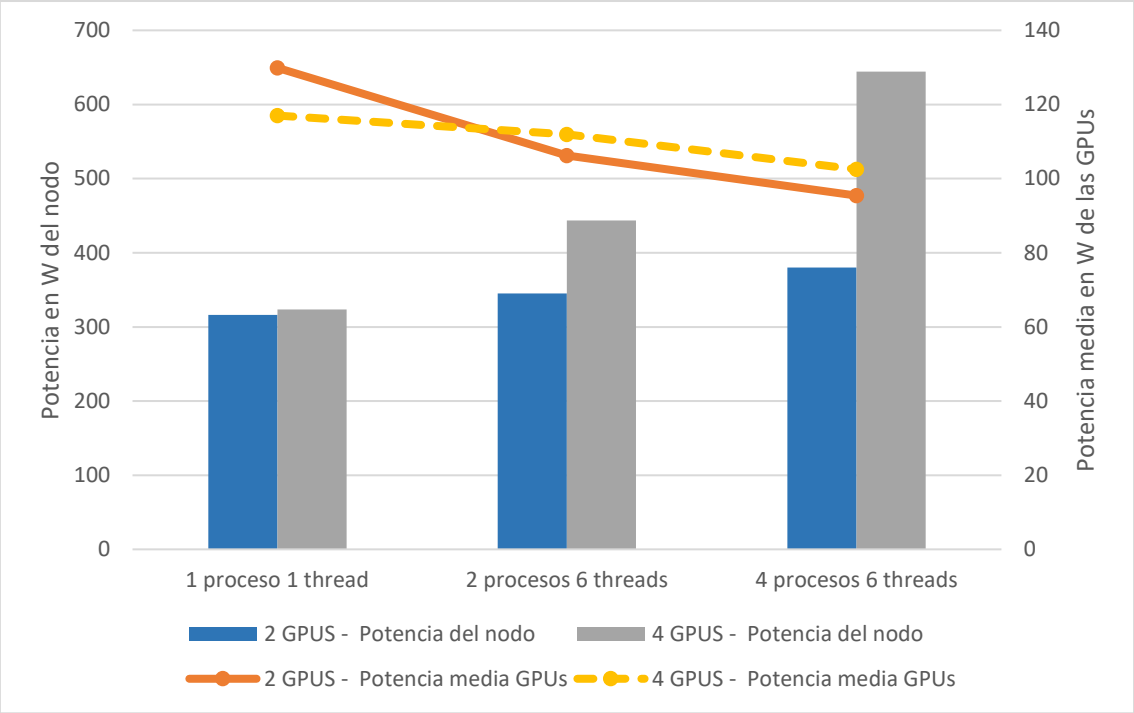


*Gráfico 21: potencia en W en MVAPICH2 con múltiples GPUs*

De las gráficas se llega a la conclusión de que la mejor configuración para un solo nodo que utiliza múltiples GPUs es la de 4 procesos y 6 threads en 4 GPUs, puesto que cada uno de los procesos está siendo ejecutado por una GPU en exclusividad.

### 5.2.4. Comparativa de memoria distribuida

En esta sección se pretende realizar una comparativa de las pruebas de memoria distribuida en la medida de lo posible debido a las partes comunes entre ellas, puesto que en algunas pruebas han intervenido variables que en otras pruebas no lo han hecho. Así pues, se ha escogido para la comparativa la ejecución de Gromacs con un número creciente de nodos, donde la configuración final sea de 2 procesos MPI por nodo y 6

threads por cada proceso OpenMP, utilizando además aceleración por GPU con NVML. Así pues, el resultado de la comparativa se muestra a continuación:

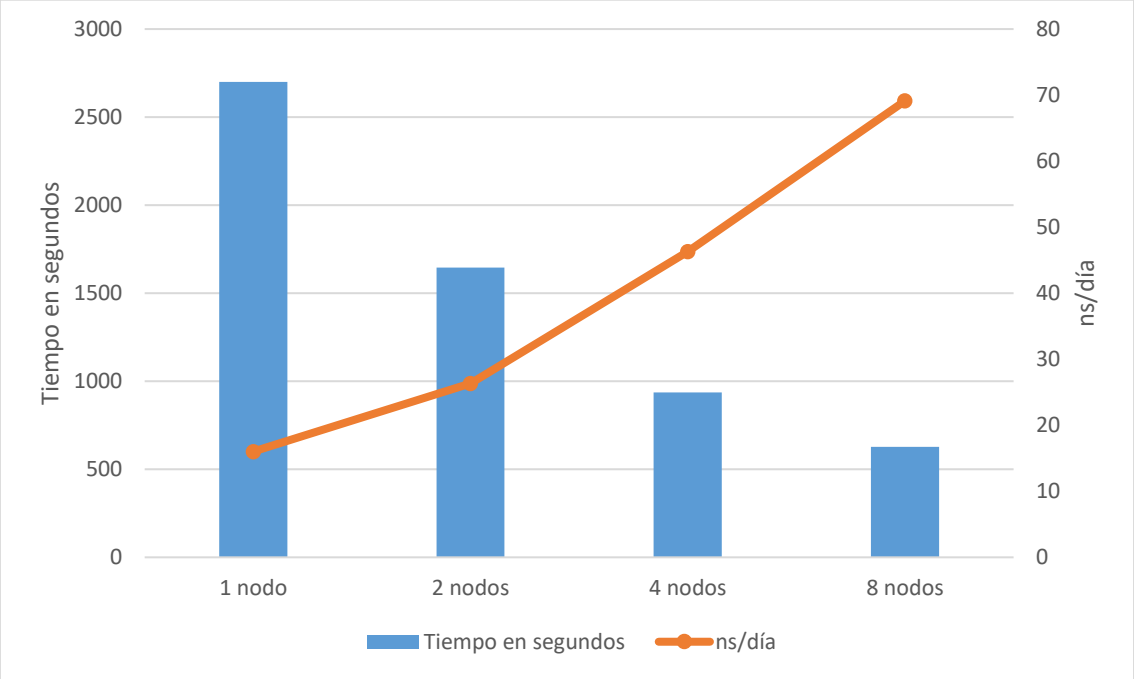


*Gráfico 22: primera comparativa memoria distribuida*

Como puede observarse, a la misma configuración de procesos y threads, el rendimiento aumenta significativamente a medida que aumenta el número de nodos empleados. De forma evidente se aprecia que la mejor configuración es con 8 nodos, a pesar de que como demuestra el siguiente gráfico, la aceleración y eficiencia obtenidas distan mucho de tener valores ideales:

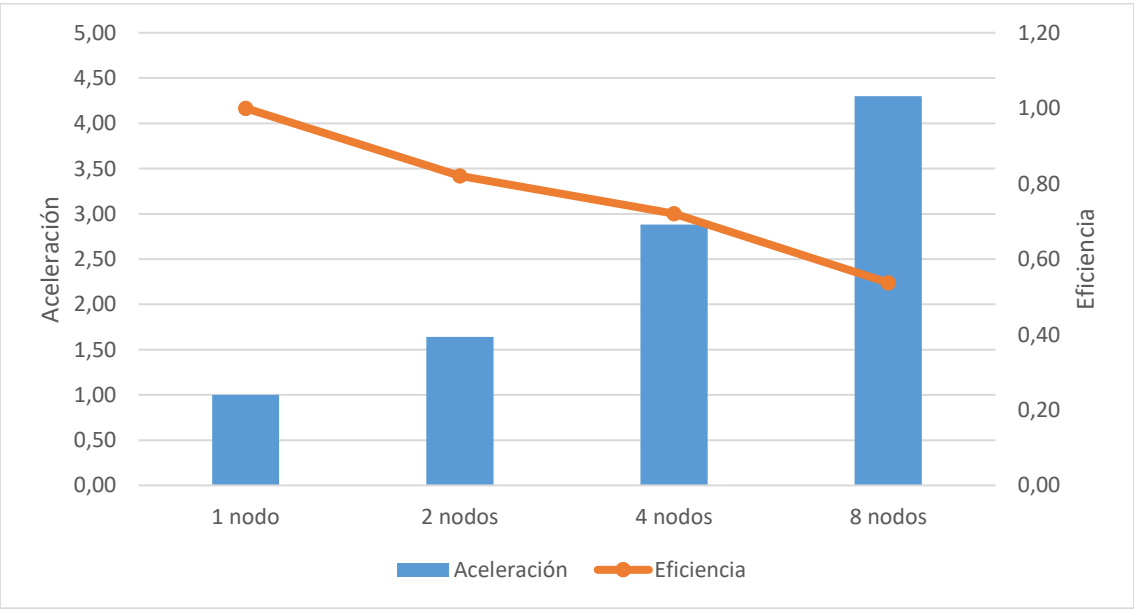


*Gráfico 23: aceleración y eficiencia obtenidas*

Por otro lado, se ha realizado otra comparativa en la que se contrasta el rendimiento obtenido usando MVAPICH2 con múltiples GPUs en un solo nodo. Para ello, contamos con la configuración de 2 procesos MPI y 6 threads por proceso. El resultado es el representado en el gráfico 22, en el que se observa que el rendimiento aumenta de forma evidente cuando se pasa de 1 a 2 GPUs pero, sin embargo, el rendimiento se mantiene constante cuando se pasa de 2 a 4 GPUs. Este resultado es debido a que para el caso de 1 GPU, los 2 procesos comparten la misma GPU, algo que no ocurre en el caso de 2 GPUs, ya que cada proceso es ejecutado por una GPU distinta. Sin embargo, en el caso de 4 GPUs, 2 de ellas están inactivas al no haber procesos que sean ejecutados por ellas.

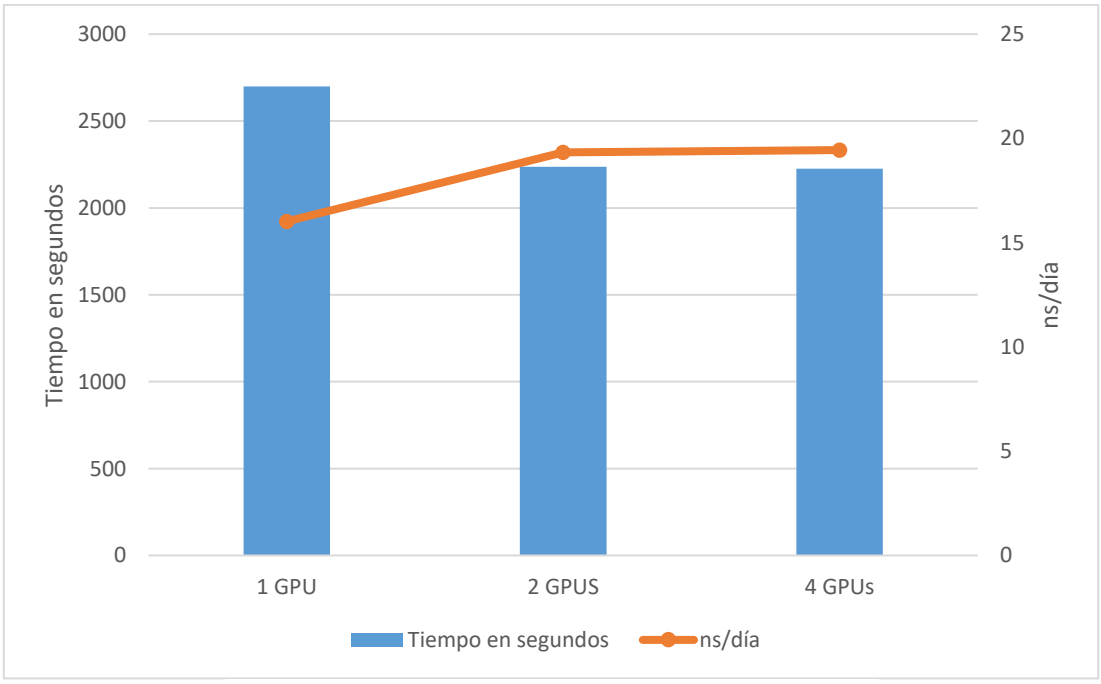


*Gráfico 24: segunda comparativa memoria distribuida*

Para concluir, se ha diseñado una última prueba para conocer qué configuración de entre las siguientes obtiene mejor rendimiento, un único nodo con 4 GPUs o 4 nodos con una sola GPU cada nodo. Para ello, se ha diseñado una sencilla prueba en la que se ejecuta el test *ion_channel* de Gromacs con 200.000 pasos de las dos formas siguientes:

- Memoria compartida en un nodo con 4 GPUs, 4 threads MPI y 3 threads OpenMP por cada thread MPI
- Memoria distribuida con MVAPICH2 con 4 nodos, 1 GPU cada nodo, 4 procesos totales y 3 threads OpenMP por cada proceso MPI

El resultado se muestra en la siguiente gráfica, en la que se llega a la conclusión de que, si el administrador tuviera la posibilidad de ejecutar Gromacs en ambas configuraciones, debería optar por la opción de 1 nodo y 4 GPUs si desea alcanzar las máximas prestaciones.

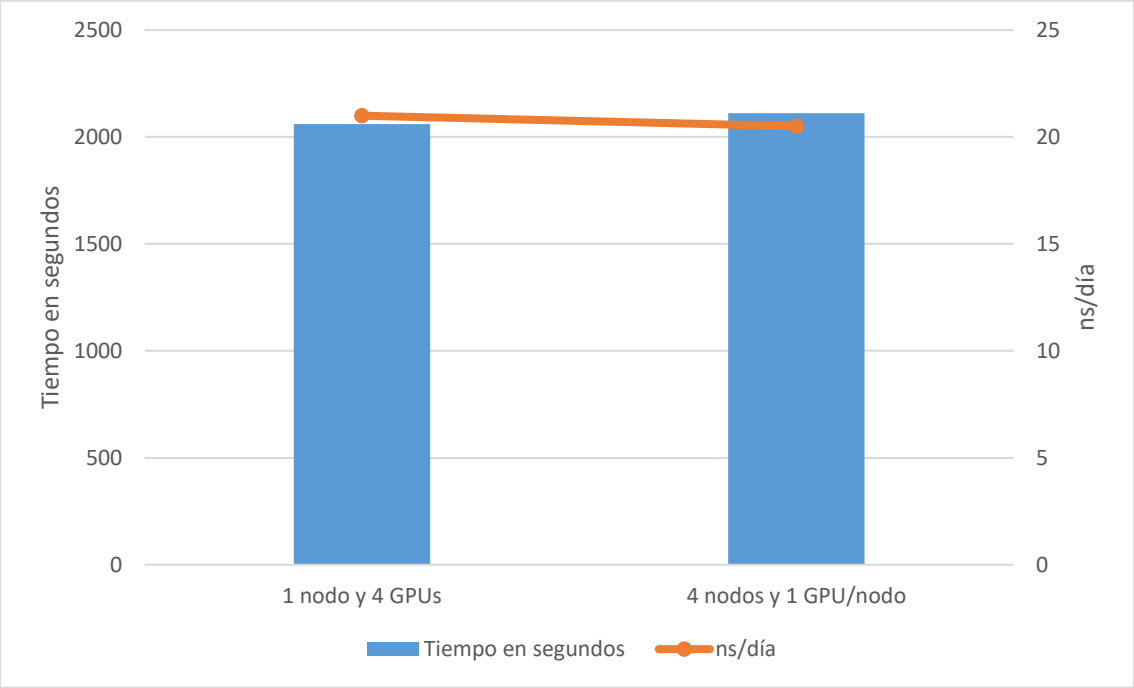


*Gráfico 25: tercera comparativa memoria distribuida*

# 6. Conclusiones

Después de haber realizado la batería de pruebas mostrada en este trabajo, se ha llegado a las siguientes conclusiones relativas a este trabajo y a la posible futura aplicación del mismo.

Dado que el objetivo de este Trabajo Fin de Grado es la realización de una simulación de administrador de sistemas que administra un clúster de altas prestaciones que ejecuta aplicaciones paralelas distribuidas, el principal resultado alcanzado pasa por haber conocido empíricamente la mejor configuración para cada una de las pruebas realizadas, las cuales son representativas de un escenario real en el que un conjunto de usuarios ejecuta una aplicación paralela distribuida.

Así pues, con los datos obtenidos en este trabajo, el administrador está capacitado para asistir técnicamente a usuarios que ejecutan este tipo de aplicaciones en un clúster de altas prestaciones, usuarios que de otro modo no tendrían posibilidad alguna de alcanzar las métricas adecuadas en la ejecución de estas aplicaciones. Como ya se ha comentado, el colectivo de usuarios que ejecuta aplicaciones de este tipo no necesariamente posee formación en este campo, por lo que podrían esperar del administrador encargado de las instalaciones que les asista técnicamente, y les proporcione las indicaciones para ejecutar sus aplicaciones de la forma más eficiente posible.

Como principal aspecto positivo de la simulación realizada, cabe destacar la experiencia adquirida en la gestión y administración de clústeres de alto rendimiento y, por consiguiente, de computadores de propósito general. Esta experiencia resultará sin duda alguna de aplicación futura en la vida profesional del estudiante.

Y como aspecto negativo a considerar, cabría mencionar el deseo de haber realizado un mayor número de pruebas con un número sensiblemente mayor de aplicaciones, con objeto de alcanzar una vista más amplia sobre el campo en el que se ha ubicado este trabajo.

Como último punto a mencionar, cabe destacar que el fruto obtenido con la realización de este trabajo resulta de una gran valía para el estudiante, además de haber supuesto una considerable satisfacción personal.

# 7. Anexo

En esta sección se describen de forma breve y esquemática los elementos y las tecnologías de las que se ha hecho uso durante la elaboración de este trabajo.

- **Clúster de computadores:** conjunto de computadores independientes y conectados mediante una red de alta velocidad que se comportan como si fueran un único computador. Un clúster tiene la capacidad de realizar una serie de tareas que, de realizarlas un único computador individual, emplearía un tiempo variablemente mayor.

- **Gromacs:** aplicación de alto rendimiento, multiplataforma y de código abierto empleada para realizar simulaciones de la dinámica molecular de sistemas con varios cientos de millones de partículas.

- **CPD o Centro de Procesamiento de Datos:** también llamado centro de cálculo o simplemente centro de datos, es el emplazamiento donde se ubican los recursos necesarios para el procesamiento de información de una organización. Generalmente cuenta con racks donde se ubican los componentes electrónicos, tales como servidores y elementos de comunicaciones.

- **Rack:** armario ubicado en un CPD donde se alojan los componentes electrónicos diseñados para ser alojados en un rack. Está constituido por un número variable de unidades rack. El objetivo de instalar racks es poder disponer de un número elevado de configuraciones hardware sin necesidad de ocupar demasiado espacio.

- **Unidad rack:** una unidad rack, o simplemente "U", es una unidad de medida para describir la altura de un componente diseñado para ser alojado en un rack.

- **Microprocesador:** es el componente central de un computador y el circuito integrado más complejo de un sistema informático. Inicialmente, su rendimiento se medía en base a su frecuencia de funcionamiento, pero con la aparición de los procesadores multicore, esta forma de comparación ha quedado obsoleta.

- **Procesador multinúcleo:** también llamados multicore, estos procesadores son aquellos que poseen dos o más núcleos físicos dentro del mismo circuito integrado. De esta forma, es posible obtener cierto nivel de paralelismo usando un único circuito integrado. Con la aparición de la tecnología HyperThreading de Intel, se hace necesario diferenciar entre núcleo físico y núcleo lógico.

- **Núcleo físico:** es el componente hardware por el cual un procesador multinúcleo está constituido. Así pues, un microprocesador Dual-Core dispondrá de dos núcleos físicos.

- **Núcleo lógico:** es un componente virtual generado con la tecnología HyperThreading de Intel que simula la existencia de dos procesadores en un único núcleo físico. De esta forma, un microprocesador Dual-Core con HyperThreading habilitado dispondrá de dos núcleos físicos y dos núcleos lógicos por cada núcleo físico, por lo que en total dispondrá de cuatro núcleos lógicos.

- **HyperThreading:** tecnología propietaria de la empresa Intel que simula la existencia de procesadores virtuales dentro de los procesadores físicos de un microprocesador multinúcleo. El nivel de paralelismo alcanzado con esta tecnología es a nivel de thread.

- **Hilo de ejecución o thread:** mínima unidad de procesamiento que puede ser planificada por un sistema operativo. La generación de un nuevo thread permite a una aplicación realizar múltiples tareas de forma concurrente, siempre y cuando el hardware encargado de realizar estas tareas esté preparado para ello. Por su naturaleza, un hilo de ejecución forma parte de un proceso.

- **Proceso:** conjunto formado por una serie de recursos informáticos y por uno o varios hilos de ejecución que comparten esos mismos recursos. Dado que estos recursos son compartidos por todos los threads, todos ellos tienen acceso total a los mismos, y cuando uno de ellos modifica un recurso, el resto de threads pueden acceder a ese dato modificado de forma inmediata.

- **Paralelismo:** paradigma de la computación en la que pueden realizarse varias tareas de forma simultánea. Existen distintos tipos de paralelismo en función de los elementos informáticos que se paralelizan, entre los que destacamos el paralelismo a nivel de instrucción, de datos y de tareas.

- **Computación paralela:** tipo de computación que emplea el paralelismo y que es usada ampliamente en clústeres de alto rendimiento.

- **Ley de Amdahl:** ley matemática que establece un límite teórico máximo en la mejora global de un sistema informático cuando una parte de este sistema es mejorado. En particular, determina que la mejora obtenida en el rendimiento de un sistema debido a la mejora de un componente está limitada por la fracción de tiempo que se utiliza dicho componente. Este principio afecta directamente a la

computación de altas prestaciones, pues en efecto, dotar a un computador con el doble de su potencia inicial no implicará necesariamente que su rendimiento final sea mejorado al doble.

- **GPU o unidad de procesamiento gráfico:** microprocesador presente en las tarjetas gráficas capaz de ejecutar un alto nivel de paralelismo. Véase punto 2.2.

- **Ethernet:** estándar de redes de área local y principal protocolo para la transmisión de datos en la capa de enlace del modelo OSI. Es usado ampliamente en la computación de propósito general y también en la de altas prestaciones para la administración de los nodos que forman un clúster. A pesar de que la comunicación entre los nodos suele ser mediante una red de alta velocidad como InfiniBand, puede usarse Ethernet para el mismo propósito.

- **CUDA:** librería propietaria de Nvidia que permite la programación de aplicaciones que usen la capacidad de paralelismo de las GPUs. Véase punto 2.2.

- **Memoria compartida:** tipo de memoria que puede ser accedida por múltiples programas con objeto de compartir datos entre ellos. En el contexto de la computación de altas prestaciones, la memoria compartida suele hacer referencia a un programa que se ejecuta en un único nodo mediante un número variable de hilos de ejecución.

- **Memoria distribuida:** tipo de memoria empleada en la computación de altas prestaciones en la que los nodos de un clúster de computadores comparten datos mediante el paso de mensajes. En este contexto, la memoria distribuida hace referencia a un programa que se está ejecutando concurrentemente entre varios nodos de un clúster, independientemente de que cada uno de los nodos esté utilizando memoria compartida de forma individual.

- **openMP:** interfaz de programación de aplicaciones para la programación multiproceso usando memoria compartida. Mediante el uso de directivas openMP aplicadas directamente sobre el código fuente, es posible dotar a los programas de la capacidad de ejecutarse concurrentemente, siempre y cuando el hardware subyacente disponga de los recursos necesarios.

- **Thread-MPI:** librería que implementa el estándar de paso de mensajes MPI y que se encuentra en Gromacs que permite el uso de MPI en ejecuciones de memoria compartida. Véase punto 2.5.

- **MPI:** estándar que define la sintaxis y semántica de las funciones que implementan las librerías de paso de mensajes entre computadores. Véase punto 2.4.
- **InfiniBand:** red de alta velocidad empleada generalmente en clústeres de altas prestaciones. Véase 2.3.
- **openMPI:** librería que implementa el estándar MPI y que es ampliamente usada en computación de altas prestaciones. Permite al programador desarrollar programas que se ejecutan de forma concurrente por los distintos nodos de un clúster de computadores mediante el paso de mensajes.
- **MVAPICH2:** librería que, al igual que openMPI, implementa el estándar MPI y permite al programador desarrollar aplicaciones que se ejecutan concurrentemente por los nodos de un clúster mediante el paso de mensajes entre los distintos nodos.
- **SSH:** protocolo de acceso a máquinas remotas que permite el manejo de las mismas de forma segura mediante un intérprete de comandos. Permite al usuario autenticarse en la máquina remota mediante contraseñas y mediante la gestión de claves públicas. Asimismo, todo el tráfico existente entre los terminales conectados por una conexión SSH permanece siempre cifrado.
- **NVML (nVidia Management Library):** interfaz de programación de aplicaciones propietaria de nVidia que posibilita la monitorización y gestión directa de las GPUs nVidia. Véase 2.2.

# 8. Bibliografía

[1] GROMACS Depelopment Team. “Gromacs”. [En línea]. Disponible en: http://www.gromacs.org/. [Última consulta: 25 de agosto de 2016].

[2] NVIDIA Corporation. “Qué es el GPU Computing”. [En línea]. Disponible en: http://www.nvidia.es/object/gpu-computing-es.html. [Última consulta: 25 de agosto de 2016].

[3] NVIDIA Corporation. “Qué es el GPU Computing”. [En línea]. Disponible en: http://www.nvidia.es/object/gpu-computing-es.html. [Última consulta: 25 de agosto de 2016].

[4] NVIDIA Corporation. “NVIDIA Management Library (NVML)”. [En línea]. Disponible en: https://developer.nvidia.com/nvidia-management-library-nvml. [Última consulta: 25 de agosto de 2016].

[5] Khronos Group. “The open standard for parallel programming of heterogeneous systems”. [En línea]. Disponible en: https://www.khronos.org/opencl. [Última consulta: 25 de agosto de 2016].

[6] Cyberseguridad.net. “¿Qué es InfiniBand?”. [En línea]. Disponible en: http://cyberseguridad.net/index.php/280-que-es-infiniband. [Última consulta: 25 de agosto de 2016].

[7] Wikipedia.org. “Interfaz de paso de mensajes”. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes. [Última consulta: 25 de agosto de 2016].

[8] The Open MPI Project. “Open MPI: Open Source High Performance Computing”. [En línea]. Disponible en: https://www.open-mpi.org. [Última consulta: 25 de agosto de 2016].

[9] The Ohio State University. “MVAPICH”. [En línea]. Disponible en: http://mvapich.cse.ohio-state.edu. [Última consulta: 25 de agosto de 2016].

[10] GROMACS Depelopment Team. “Acceleration and parallelization”. [En línea]. Disponible en: http://www.gromacs.org/Documentation/Acceleration_and_parallelization. [Última consulta: 25 de agosto de 2016].

[11] GROMACS Development Team. “Gromacs Documentation”. [En línea]. Disponible en: http://manual.gromacs.org/documentation/5.1.2/index.html. [Última consulta: 25 de agosto de 2016].

[12] Open MPI Team. “mpirun(1) man page”. [En línea]. Disponible en: https://www.open-mpi.org/doc/v2.0/man1/mpirun.1.php. [Última consulta: 25 de agosto de 2016

[13] MVAPICH Team. “MVAPICH 2.0 User Guide”. [En línea]. Disponible en: http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html. [Última consulta: 25 de agosto de 2016].