Evaluación De Contenedores De Docker Para Aplicaciones Sobre GPU

Gissel Daniela León Godoy

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2020

Evaluación De Contenedores De Docker Para Aplicaciones Sobre GPU

Gissel Daniela León Godoy

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
Sergio Alberto Abreo Carrillo
Doctorado en Ingeniería

Co-Director
Ana Beatriz Ramirez Silva
Ph.D. en Ing. Eléctrica.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2020

A Andrés Manjarrés, mi amigo, mi pareja, por no solo 5 años de una relación inolvidable sino también incomparable, apoyándonos dentro y fuera de la universidad en todo momento y superando cualquier adversidad, él es quien ahora, camina conmigo para toda la vida.

A Liliana, mi querida mamá, una mujer luchadora, capaz de todo lo que se propone, que siempre estuvo dispuesta a apoyarme cuando lo necesitaba, te amo má.

A mi familia siempre atenta y orgullosa de mis logros, mis amigas Paula, Ana y los indignados que siempre permanecimos juntos en todo.

A todos los amigos que hice en la universidad, cada uno aportó algo para mi crecimiento personal y espero haber hecho lo mismo en ustedes, los recordaré siempre.

Universidad Industrial de Santander

SISTEMA DE TRABAJOS DE GRADO
ADMINISTRACIÓN DE TRABAJOS DE GRADO

Fecha Impresión:
21 febrero 2020

Codigo:	21581	**Fecha**	30-ago-2019
Titulo:	EVALUACIÓN DE CONTENEDORES DE DOCKER PARA APLICACIONES SOBRE GPU.		
Nota Proyecto:	4.6	**Fecha Registro**	21-feb-2020
Estado:	APROBADO		
Tipo Trabajo:	TRABAJO DE INVESTIGACION		

Estudiantes

Código	Nombre	Programa Académico
2141848	LEON GODOY GISSEL DANIELA	26-INGENIERIA ELECTRONICA

Directores

Documento	Nombre	Clase	Firma
C-91522275	SERGIO ALBERTO ABREO CARRILLO	DIRECTOR	
C-63527848	ANA BEATRIZ RAMIREZ SILVA	CODIRECTOR	

Calificadores

Documento	Nombre	Firma
C-91533242	WILLIAM ALEXANDER SALAMANCA BECERRA	
C-1098696549	JHEYSTON OMAR SERRANO LUNA	

ENTREGA DE TRABAJOS DE GRADO, TRABAJOS DE INVESTIGACION O TESIS Y AUTORIZACIÓN DE SU USO A FAVOR DE LA UIS

Yo, **Gissel Daniela León Godoy** , mayor de edad, vecino de Bucaramanga, identificado con la Cédula de Ciudadanía No. **1095833073** de **Floridablanca**, actuando en nombre propio, en mi calidad de autor del trabajo de grado, del trabajo de investigación, o de la tesis denominada(o):

Evaluación de contenedores de Docker para aplicaciones sobre GPU,

hago entrega del ejemplar respectivo y de sus anexos de ser el caso, en formato digital o electrónico (CD o DVD) y autorizo a LA UNIVERSIDAD INDUSTRIAL DE SANTANDER, para que en los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia, utilice y use en todas sus formas, los derechos patrimoniales de reproducción, comunicación pública, transformación y distribución (alquiler, préstamo público e importación) que me corresponden como creador de la obra objeto del presente documento. PARÁGRAFO: La presente autorización se hace extensiva no sólo a las facultades y derechos de uso sobre la obra en formato o soporte material, sino también para formato virtual, electrónico, digital, óptico, uso en red, Internet, extranet, intranet, etc., y en general para cualquier formato conocido o por conocer.

EL AUTOR – ESTUDIANTE, manifiesta que la obra objeto de la presente autorización es original y la realizó sin violar o usurpar derechos de autor de terceros, por lo tanto, la obra es de su exclusiva autoría y detenta la titularidad sobre la misma. PARÁGRAFO: En caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, EL AUTOR / ESTUDIANTE, asumirá toda la responsabilidad, y saldrá en defensa de los derechos aquí autorizados; para todos los efectos la Universidad actúa como un tercero de buena fe.

Para constancia se firma el presente documento en dos (02) ejemplares del mismo valor y tenor, en Bucaramanga, a los 25 días del mes de Febrero de Dos Mil Veinte 2020.

EL AUTOR / ESTUDIANTE:

Gissel Daniela León Godoy

AGRADECIMIENTOS

Agradezco al profesor Sergio Abreo, por su acompañamiento en la dirección de este proyecto, como profesor y amigo, depositando confianza y al mismo tiempo el apoyo necesario para obtener los mejores resultados en este trabajo.

Igualmente a la profesora Ana Ramírez, por su comprensión y cordialidad a la hora de reunirnos.

También a Paola Hernandez y María Fernanda Rueda por su apoyo en las tareas administrativas y de forma personal siendo buenas amigas al interior del grupo CPS.

No solo venimos a estudiar a la universidad, sino a aprender todo lo que significa la vida, crecer y ser ciudadanos activos, personas que luchan por lo que es correcto y también abrazan las diferencias de los demás, agraezco a la UIS por ese aporte personal en cada estudiante.

CONTENIDO

LISTA DE FIGURAS

pág.

[1] Mark HARRIS. *How to Optimize Data Transfers in CUDA C/C++*. Ed. por Nvidia. `https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/`. 2012.

[2] Mark HARRIS. *How to Overlap Data Transfers in CUDA C/C++*. Ed. por Nvidia. `https://devblogs.nvidia.com/how-overlap-data-transfers-cuda-cc/`. 2012.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: EVALUACIÓN DE CONTENEDORES DE DOCKER PARA APLICACIONES SOBRE GPU [*]

AUTOR: GISSEL DANIELA LEÓN GODOY [**]

PALABRAS CLAVE: Docker, Rendimiento, CUDA, Transferencia de Datos, Nvdia-Docker.

DESCRIPCIÓN:

En el mundo de la investigación científica cuando se obtienen nuevos resultados es necesario que después de obtenerlos estos puedan corroborarse en cualquier otra situación o en cualquier otro lugar (reproducibilidad y repetibilidad). Los contenedores de Linux se presentan como una opción para solucionar el problema de reproducibilidad en software científico, ya que muchas veces la tarea de hacer un producto compatible con diferentes entornos de trabajo se hace casi imposible. El objetivo principal de esta investigación es evaluar el rendimiento de Docker sobre GPUs, las cuales son utilizadas ampliamente en investigaciones y responder a la pregunta ¿se obtiene algún beneficio o pérdida al usar un contenedor de Linux con Docker para desplegar aplicaciones de alto rendimiento?

Para visualizar el comportamiento del contenedor, se realizaron diferentes pruebas y se evaluaron principalmente tiempos de ejecución y compilación. Las pruebas se hicieron sobre sistema nativo y usando contenedores con el fin de comparar los dos escenarios y evaluar el rendimiento del contenedor. Con este trabajo se aporta más evidencia de que los contenedores son una herramienta importante, que no ofrece pérdidas considerables y que su uso en la comunidad científica debe crecer con el fin de aumentar la reproducibilidad y repetibilidad de la investigación científica..

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Sergio Alberto Abreo Carrillo Doctor en Ingeniería. Co-Director: Ana Beatríz Ramírez Silva PhD. en Ing. Eléctrica.

ABSTRACT

TITLE: EVALUATION OF DOCKER CONTAINERS FOR GPU APPLICATIONS [*]

AUTHOR: GISSEL DANIELA LEÓN GODOY [**]

KEYWORDS: Docker, Performance, CUDA, Data Transfer, Nvdia-Docker.

DESCRIPTION:

In the scientific investigation world when we obtain some result is necessary that this kind of results could be corroborated in any place or any situation, that is the meaning or do something repeatable and reproducible. The Linux Containers appears like an option to solve the problem of reproducibility in scientific software because so many times the job to do a compatible product with different environments is almost impossible. The principal object in this research work is to observe how is the behavior in the Docker containers and evaluate the performance of Docker on GPUs (The GPUs are widely used in scientific investigations). Here we have to answer a big question and is if we obtain a benefit or loss when we use a Linux Container with Docker to deploy or execute a high-performance application.

To see the behavior of the container, we made different tests and we note that the most important characteristic to evaluate it was the time. We took the compilation time and the execution time in every trial; it happens in the native system and in the same way using the Docker containers to compare these two scenarios and evaluate the performance of the container.

In this thesis, we are looking to make a contribution providing evidence about the containers to say that it was an important tool that does not offer significant losses and the use of it in the scientific community may grow to increase the reproducibility and repeatability of the scientific investigation.

---

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Sergio Alberto Abreo Carrillo Doctor en Ingeniería. Co-Director: Ana Beatríz Ramírez Silva PhD. en Ing. Eléctrica.

INTRODUCCIÓN

El desarrollo científico y en general las aplicaciones de alta demanda computacional se han beneficiado del hardware para computación de alto desempeño como GPUs con grandes cantidades de núcleos y CPUs especializadas con altas frecuencias, bajos consumos de potencia, gran capacidad en la memoria interna, entre otros. Todo esto, en forma de *cluster*, hace que cada día el desarrollo científico pueda avanzar más y ser más eficiente. Sin embargo, un aspecto importante cuando se habla del desarrollo computacional es realizar algoritmos reproducibles y repetibles, los cuáles ofrezcan los mismos resultados bajo las mismas condiciones. En este punto es cuando se entra en conflicto ya que debido al amplio rango de opciones para crear un entorno de trabajo, éste puede no ser compatible con otros y terminar perdiendo su capacidad de portabilidad.

Por esta razón los contenedores empiezan a jugar un papel importante, pese a que su uso está más enfocado en el área del desarrollo de aplicaciones comerciales. Los contenedores son una tecnología que permite encapsular todo un entorno de trabajo y hacerlo portátil solucionando los problemas de compatibilidad y la necesidad de descargar las librerías o paquetes necesarios para ejecutar una aplicación. Esto surge como una alternativa más eficiente a las máquinas virtuales que solucionaban estos mismos problemas pero que hacían uso de una gran cantidad de recursos, ralentizando los procesos.

Cuando se habla de contenedores, sin lugar a duda el gran referente en la industria es la tecnología de contenedores de Docker, el cual cuenta con una gran biblioteca de imágenes ya predefinidas para desplegar contenedores. Entre las imágenes que tiene Docker se encuentra Nvidia-Docker la cual permite programar GPUs de

la compañía Nvidia utilizando el lenguaje de programación CUDA C, el cual es un lenguaje basado en C pero con librerías específicas para aprovechar la paralelización que ofrecen la GPUs. En este tipo de contenedores es donde se enfoca este trabajo, analizando qué sucede con el rendimiento de las aplicaciones desarrolladas en CUDA cuando se implementan utilizando un contenedor y comparando los resultados cuando se hace en el entorno nativo del equipo.

El trabajo está organizado de la siguiente forma: la teoría referente a Docker y su importancia en este trabajo es expuesta en el capítulo 3.1, la descripción de las pruebas realizadas y los resultados obtenidos se presentan en el capítulo 4.1 y finalmente se hace una conclusión de trabajo y se presentan las posibilidades de futuros trabajos en los capítulos 6.

OBJETIVOS

Objetivo general

- Evaluar el rendimiento de contenedores en Linux para aplicaciones sobre GPUs.

Objetivos específicos

- Implementar un contenedor de linux utilizando herramientas como Docker sobre GPUs.
- Seleccionar métricas para la medición de rendimiento del contenedor elegido.
- Validar el desempeño del contenedor por medio de un conjunto de aplicaciones de HPC.

Marco Teórico

Docker

Docker es una plataforma de código abierto que ha venido tomando fuerza en la comunidad de desarrollo de software en los últimos años. Esta plataforma hace uso de la tecnología de contenedores para ofrecer la posibilidad de empaquetar todo un entorno de desarrollo y que sea totalmente aislado. Esto produce que no se necesite realizar toda una instalación de paquetes y librerías previas a la ejecución de algún código. Además, permite la ejecución simultánea de varios de estos contenedores.

Antes de pasar a los siguientes capítulos, es necesario aclarar algunos conceptos importantes respecto a Docker. Un contenedor es una virtualización del sistema operativo, el *kernel* se comparte con el equipo *Host* donde se despliega el contenedor. Para construir estos contenedores se hace uso de una *imagen* que describe todo el sistema de archivos que ejecutará el contenedor. Por ejemplo, es posible tener imágenes de Ubuntu 16.04 lo que nos permitiría desplegar uno o varios contenedores de esta distribución de Linux. Las imágenes que se crean se pueden almacenar en un registro (el mayor registro de imágenes de Docker es Docker Hub) el cual está conectado con Docker y permite descargar una gran cantidad de imágenes diseñadas por terceros. Las imágenes también se pueden crear desde cero o pueden surgir a partir de contenedores modificados. Continuando con el ejemplo de la imagen de Ubuntu, es posible desplegar un contenedor de Ubuntu 16.04 u otra versión e instalarle algún software específico y posteriormente construir una imagen de ese nuevo contenedor con el software instalado.

Nvidia-Docker

Nvidia, una de las empresas lideres en la producción de GPUs publicó en Docker Hub imágenes con sus herramientas previamente instaladas . Los únicos requisitos para hacer uso de estas imágenes son tener instalado Docker y contar con el *driver* apropiado para controlar la GPU.

Trabajos relacionados

Gran cantidad de trabajos se han centrado en el rendimiento de Docker vs máquinas virtuales [1] y vs el sistema nativo [2]. Estos trabajos se centran en aplicaciones para CPU. Algunos trabajos como [3] por otro lado se centran en aplicaciones específicas para GPU y evalúan varios tipos de contenedores como Docker o Singularity [4] para comparar cuál es más útil, haciendo uso de *benchmarks* estandarizados.

Debido al creciente uso de la nube para el desarrollo en GPU algunos trabajos han

---

[1] A.M. Joy. "Performance comparison between Linux containers and virtual machines". En: *Conference Proceeding - 2015 International Conference on Advances in Computer Engineering and Applications, ICACEA 2015* (2015), págs. 342-346. DOI: 10.1109/ICACEA.2015.7164727.

[2] M.G. Xavier y col. "Performance evaluation of container-based virtualization for high performance computing environments". En: *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013* (2013), págs. 233-240. DOI: 10.1109/PDP.2013.41.

[3] Carlos ARANGO, Rémy DERNAT y John SANABRIA. "Performance Evaluation of Container-based Virtualization for High Performance Computing Environments". En: *CoRR* abs/1709.10140 (2017). arXiv: 1709.10140. URL: http://arxiv.org/abs/1709.10140.

[4] Gregory KURTZER, Vanessa SOCHAT y Michael BAUER. "Singularity: Scientific containers for mobility of compute". En: *PLOS ONE* 12.5 (mayo de 2017), págs. 1-20. DOI: 10.1371/journal.pone.0177459.

explorado el rendimiento de Docker para trabajo científico desarrollado en la nube [5], [6]. Todos estos trabajos muestran que los contenedores no afectan los desarrollos en CPU y GPU, pero no se experimenta con el uso de estrategias de optimización para aplicaciones de alto desempeño necesarias cuando se trabajan problemas complejos desde el punto de vista computacional.

---

[5] Pankaj SAHA y col. "Evaluation of Docker Containers for Scientific Workloads in the Cloud". En: *CoRR* abs/1905.08415 (2019). arXiv: 1905.08415. URL: http://arxiv.org/abs/1905.08415.

[6] MINSU Chae, HWAMIN Lee y KYEOL Lee. "A performance comparison of linux containers and virtual machines using Docker and KVM". En: *Cluster Computing* 22.1 (2019), págs. 1765-1775. DOI: 10.1007/s10586-017-1511-2.

Pruebas

Para la realización de todas las pruebas se utilizó un equipo de escritorio con las especificaciones descritas en la tabla 1

Tabla 1. Características de hardware y software del equipo utilizado para ejecutar las pruebas.

Item	Modelo o Versión
GPU	GeForce GTX 750 Ti
CPU	Intel Xeon CPU E5-1650 v3@3.50GHz
Placa Base	0K240Y v.A02
Versión de CUDA	10.2
Driver CUDA	440.33.01
Version Docker	19.03.5
SO Nativo	Ubuntu 16.04
Imagen Docker	nvidia/cuda:10.2-devel-ubuntu16.04

Como se planteó anteriormente el enfoque es observar la implicación del uso de contenedores en aplicaciones que hicieran uso de GPUs. Como primera medida se utilizó una aplicación desarrollada en el grupo de investigación CPS para determinar que métricas eran relevantes para analizar. Esta aplicación luego se llevo a un contenedor con las características mostradas en la tabla 1 y se repitieron las mediciones para hacer la comparación de rendimiento.

La aplicación elegida fue la simulación de la propagación de un frente de onda elás-

tico 2D [7]. La propagación se diseñó siguiendo los parámetros descritos en la Tabla 2. Para analizar el costo computacional tanto en Docker como en el sistema nativo de la aplicación elegida se utilizó la herramienta *Nvidia profile* [8], la cuál permite medir el consumo de memoria y tiempos de ejecución (Listing 4.1 y Listing 4.2). Con esto se observó que las variaciones entre las dos pruebas se hacían visibles únicamente en los tiempos de ejecución. El consumo de memoria y la forma de ejecutar la aplicación es exactamente igual. Esto se puede deducir de la teoría de Docker ya que la cantidad de memoria que utiliza depende principalmente del algoritmo y no tanto de su forma de ejecución. Sin embargo, dependiendo de la gestión de los paquetes y de cómo se administran los recursos se puede ganar o perder tiempo al utilizar Docker.

Tabla 2. Parámetros de diseño para la prueba de la simulación de la propagación de un frente de onda elástico 2D.

Parámetro	**Valor**
Tamaño del modelo en X	600 puntos
Tamaño del modelo en Y	600 puntos
Paso espacial	5 metros
Paso temporal	1 milisegundo
Tiempo de propagación	1 segundo
Región CPML	10 puntos

Listing 4.1. Resumen de recursos utilizados para la simulación de la propagación de un frente de onda elástico 2D en el sistema nativo.

[7] Andrés MANJARRÉS. “Implementación De Un Módulo De Propagación De Onda Elástica 2D Utilizando Un Cluster De GPUS”. En: *Universidad Industrial De Santander* (2018).

[8] NVIDIA. *NVIDIA Visual Profiler*. `https://developer.nvidia.com/nvidia-visual-profiler`.

```
==27176== Nvidia profile result:
Duration      Size  Throughput  Name
119.75us  1.3733MB  11.200GB/s  [CUDA memcpy HtoD]
20.449us  1.3733MB  65.583GB/s  [CUDA memset]
117.79us        0B        0B    parameters()
50.912us        0B        0B    Bou_Param()
169.67us        0B        0B    displace_paramete()
4.6400us        0B        0B    CPML_bx_rigth()
5.6640us        0B        0B    CPML_ax_rigth()
4.7680us        0B        0B    CPML_bz_bot()
5.6640us        0B        0B    CPML_az_bot()
4.3200us        0B        0B    CPML_bz_top()
4.7680us        0B        0B    CPML_az_top()
27.233us        0B        0B    add_source()
72.192us        0B        0B    CPML_V()
326.24us        0B        0B    propagator_U()
63.873us        0B        0B    Shots()
70.880us        0B        0B    CPML_T()
433.38us        0B        0B    propagator_T()
27.008us        0B        0B    add_source()
18.625us  1.3733MB  72.006GB/s  [CUDA memset]
114.91us  1.3733MB  11.671GB/s  [CUDA memcpy DtoH]
```

Listing 4.2. Resumen de recursos utilizados para la simulación de la propagación de un frente de onda elástico 2D en Docker.

```
==19236== Nvidia profile result:
Duration      Size  Throughput  Name
119.71us  1.3733MB  11.203GB/s  [CUDA memcpy HtoD]
```

20.512us	1.3733MB	65.381GB/s	[CUDA memset]
118.05us	0B	0B	parameters()
50.560us	0B	0B	Bou_Param()
169.57us	0B	0B	displace_paramete()
4.8320us	0B	0B	CPML_bx_rigth()
5.6000us	0B	0B	CPML_ax_rigth()
4.8640us	0B	0B	CPML_bz_bot()
5.4400us	0B	0B	CPML_az_bot()
4.0000us	0B	0B	CPML_bz_top()
4.5440us	0B	0B	CPML_az_top()
26.880us	0B	0B	add_source()
72.065us	0B	0B	CPML_V()
315.30us	0B	0B	propagator_U()
63.776us	0B	0B	Shots()
70.625us	0B	0B	CPML_T()
431.04us	0B	0B	propagator_T()
27.360us	0B	0B	add_source()
19.808us	1.3733MB	67.705GB/s	[CUDA memset]
115.20us	1.33733MB	11.641GB/s	[CUDA memcpy DtoH]

Por esta razón, para medir la eficiencia de los contenedores se utilizó básicamente la métrica del tiempo, pues esta es la medida de mayor importancia al correr una prueba. Se buscó mirar cómo el tiempo se veía afectado cuando se hacía uso de contenedores y cuando se ejecutaban en el sistema nativo. Los resultados de ambos tiempos y la diferencia porcentual se muestran en la Tabla 3

Tabla 3. Tiempos de compilación y ejecución de la propagación en Docker y en el sistema nativo, mostrando la desviación estándar y diferencia porcentual generada entre ellos. Los tiempos se midieron con el comando *time* y se realizaron 60 pruebas sobre cada plataforma (Nativo y Docker) para obtener datos estadísticos.

Propagación 2D	**Docker**	**Nativo**	**Diferencia**
Compilación	0.7128±0.0108	0.6785±0.0325	**-5.1** %
Ejecución	13.2647±0.2561	12.9272±0.2318	**-2.6** %

La medida se realizó utilizando dos herramientas, la primera fue el comando *time* [9] de Linux para medir el tiempo de compilación, y se hicieron uso de comandos de *Nvidia profile* y el manejo de *Cuda Events* [10] para medir los tiempos de mayor interés dentro de la ejecución de CUDA. La selección de estas herramientas se hizo basado en que ambas ofrecen una resolución del orden de los milisegundos para observar las diferencias entre las dos implementaciones.

Luego de que se caracterizó una prueba se probó por separado algunas de las partes que mostraron algunas diferencias, como por ejemplo la comunicación entre CPU y GPU (transacción entre Host y Device).

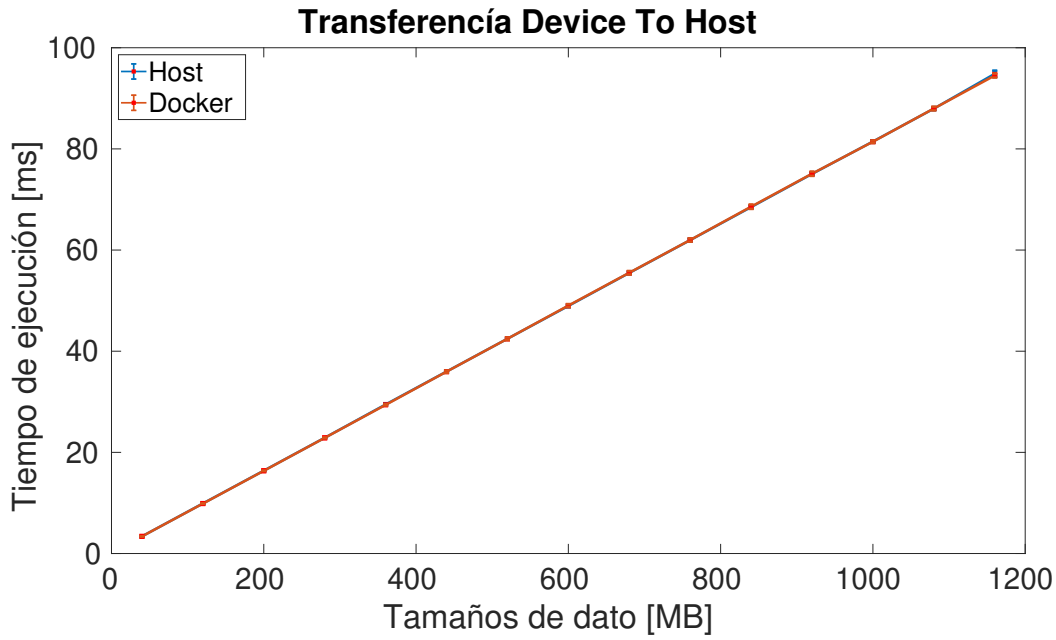
4.1.1. Transferencia *Device* to *Host* La primera prueba fue una comparativa en el tiempo que tarda Docker y el sistema nativo en realizar la transferencia de un dato de la GPU a la CPU. Para ver que tanto aumentaba el tiempo de ejecución del algoritmo se utilizaron diferentes tamaños del dato que se transfería; se hizo una variación del tamaño desde 40 MB hasta 1160 MB. Para que el valor que se obtuviera en tiempo fuera más confiable se repitieron las pruebas en 60 ocasiones

[9] Linux. *Time Command Manual*. `http://man7.org/linux/man-pages/man1/time.1.html`.

[10] Mark HARRIS. *How to Implement Performance Metrics in CUDA C/C++*. Ed. por Nvidia. `https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/`. 2012.

diferentes y se promedió su valor para dar un dato ponderado.

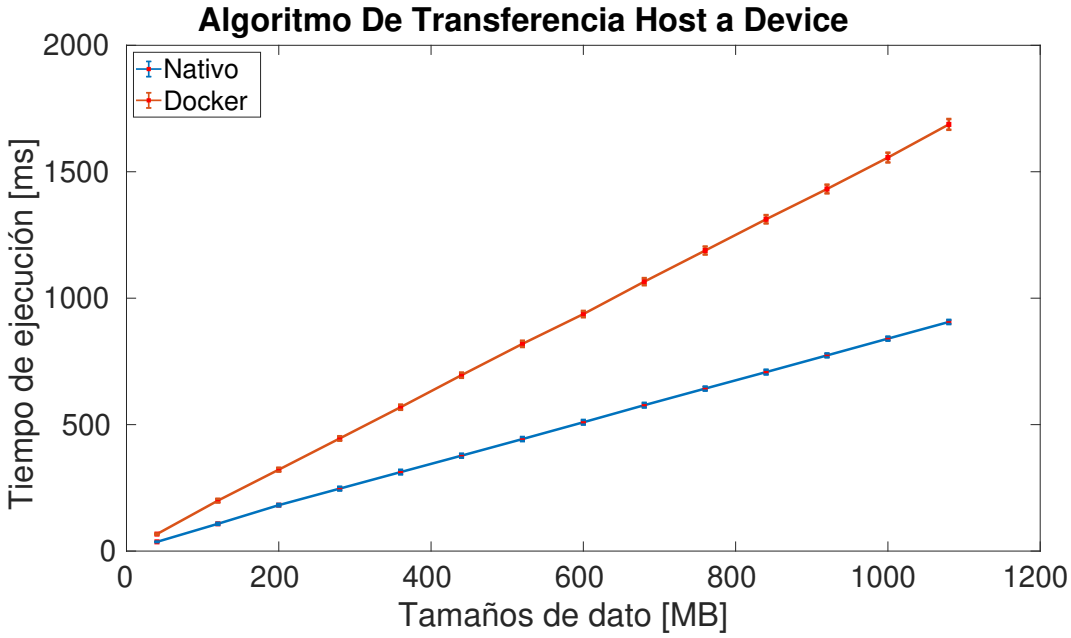
Figura 1. Prueba de Transferencia de un dato desde el *Device* al *Host*, tomando el tiempo medido en milisegundos y con una variación del tamaño del dato. Fueron medidos con la herramienta de *Nvidia profile* de CUDA.



Como se puede observar en la Figura 1 la ejecución en Docker y en el sistema nativo no presenta retrasos significativos en este tipo de transferencias.

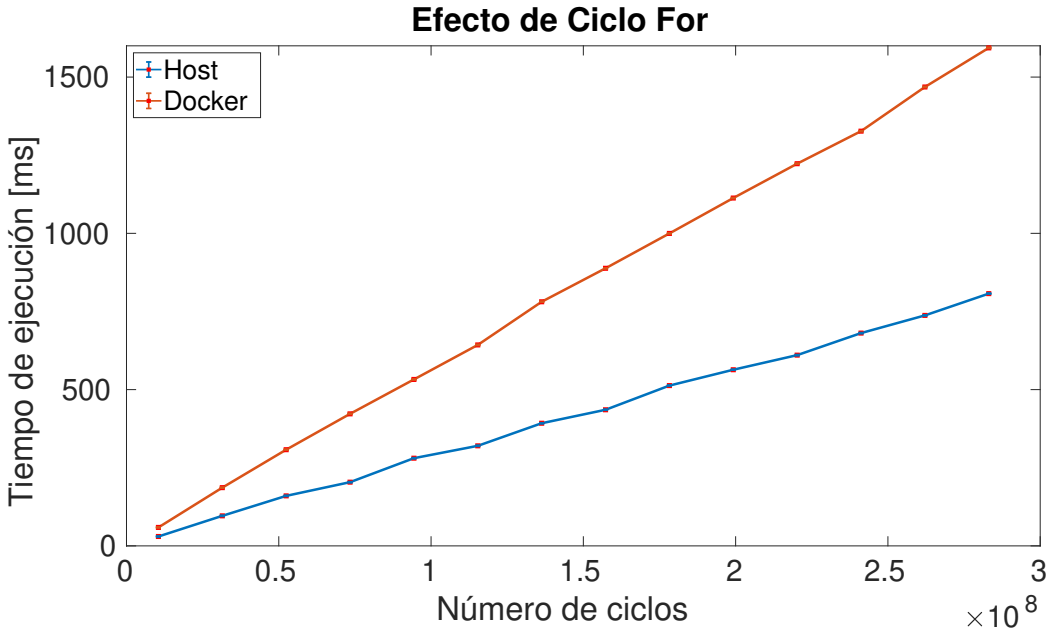
4.1.2. Transferencia *Host* to *Device* La segunda prueba fue similar a la realizada en la primera ocasión, pero esta vez una transferencia de *Host* a *Device* con la misma relación de tamaños, repeticiones y variaciones.

Figura 2. Prueba de Transferencia de un dato del *Host* al *Device*, utilizando la herramienta *Nvidia profile* de CUDA. La medida fue hecha incluyendo tanto el tiempo de la transferencia como el tiempo del ciclo previo a la transferencia.



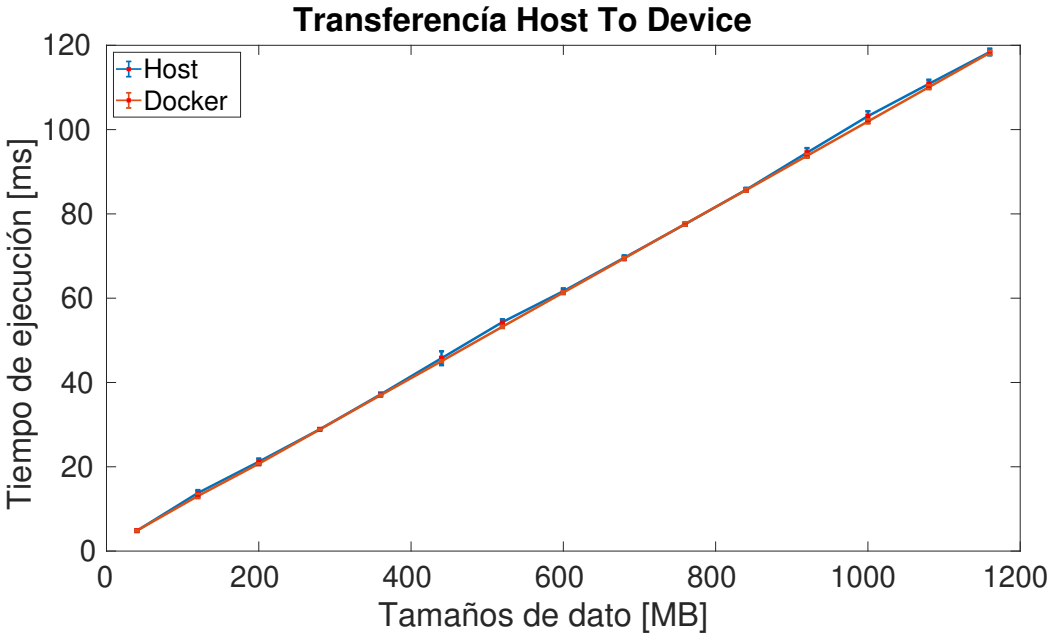
En esta prueba ocurrió algo que cabe resaltar y es que en el código se hace uso de un ciclo *for* en el *Host* para inicializar la variable a enviar. Como se observa en la Figura 2 el uso de este ciclo afecta el desempeño de la ejecución en Docker, por tanto, fue necesario realizar una prueba adicional para ver el efecto del ciclo *for* en el entorno nativo y dentro del contenedor.

Figura 3. Comparación entre los tiempos de ejecución de un ciclo *for* en el sistema Nativo y en Docker. El número de iteraciones se aumentó gradualmente de 10 millones a 283 millones y los tiempos fueron tomados utilizando las herramientas de *Nvidia profile*.



Se puede ver en la Figura 3 que al aumentar el número de iteraciones del ciclo, el *for* tiende a ser más demorado en su ejecución. Por lo mismo y tanto se midió el tiempo de ejecución de la transferencia únicamente.

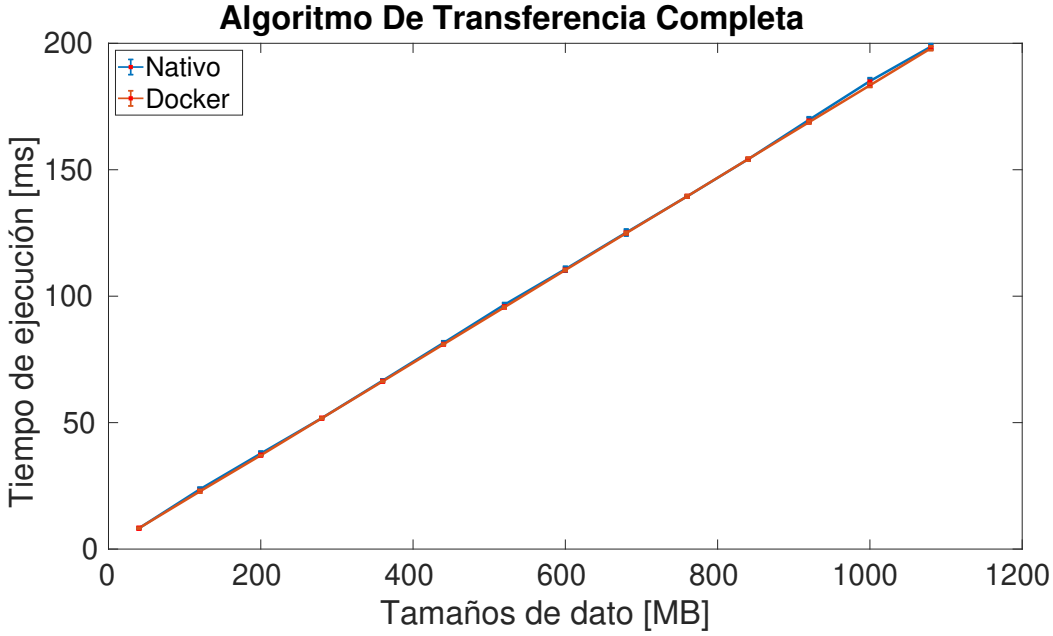
Figura 4. Prueba de Transferencia de un dato del *Host* al *Device*, utilizando la herramienta *Nvidia profile* de CUDA para medir la variación del tiempo en función del tamaño del dato.



Al analizar la transferencia unilateral de *Host* a *Device* en la Figura 4 es notable que no hay diferencias relevantes.

4.1.3. Transferencia Completa Al igual que la transferencia unilateral, en la transferencia completa se hizo una variación de tamaños para ver si se observaban cambios al probar el envió de un dato del *Host* al *Device* y luego devolverlo al *Host*.

Figura 5. Resultados de la transferencia de un dato desde el *Host* al *Device* y luego de vuelta al *Host*, variando el tamaño del dato para ver el tiempo que tarda en ejecutarse utilizando la herramienta *Nvidia profile* de CUDA.



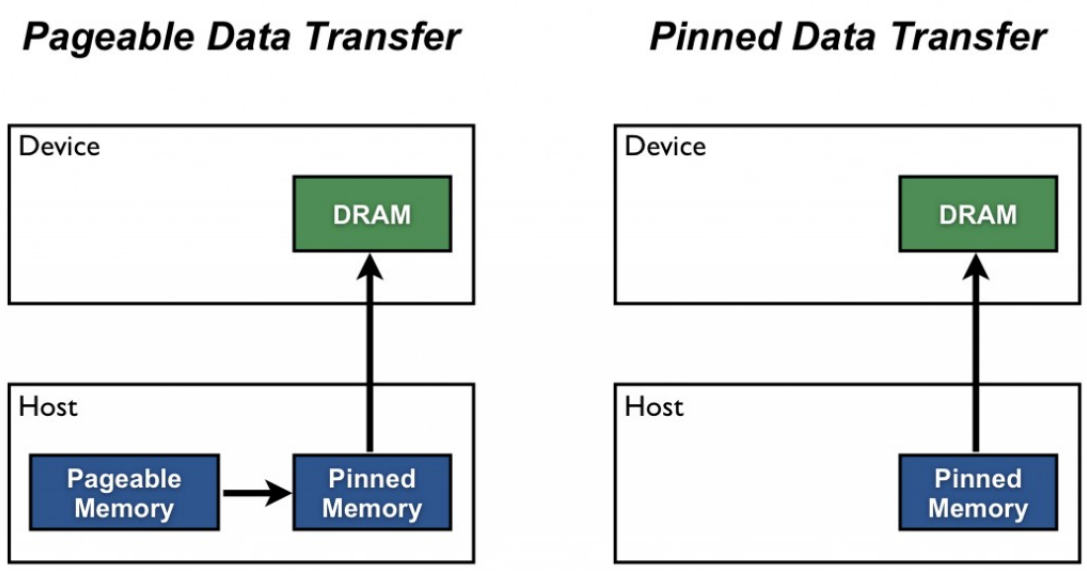
De acuerdo a lo observado en la Figura 5, realizar una transferencia completa no tiene un costo adicional si se hace en un contenedor de Docker o en el entorno Nativo.

4.1.4. *Pinned Memory* Adicionalmente, se probó si Docker soporta algunas optimizaciones y si se afectaban los tiempos por el uso del contenedor. Una de estas optimizaciones es: *pinned memory* [11]. *Pinned memory* permite realizar transferencias mucho más rápido y esto lo logra modificando el lugar donde se almacenan las variables de la CPU, como se muestra en la Figura 6. Por defecto estas variables

[11] Mark HARRIS. *How to Optimize Data Transfers in CUDA C/C++*. Ed. por Nvidia. `https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/`. 2012.

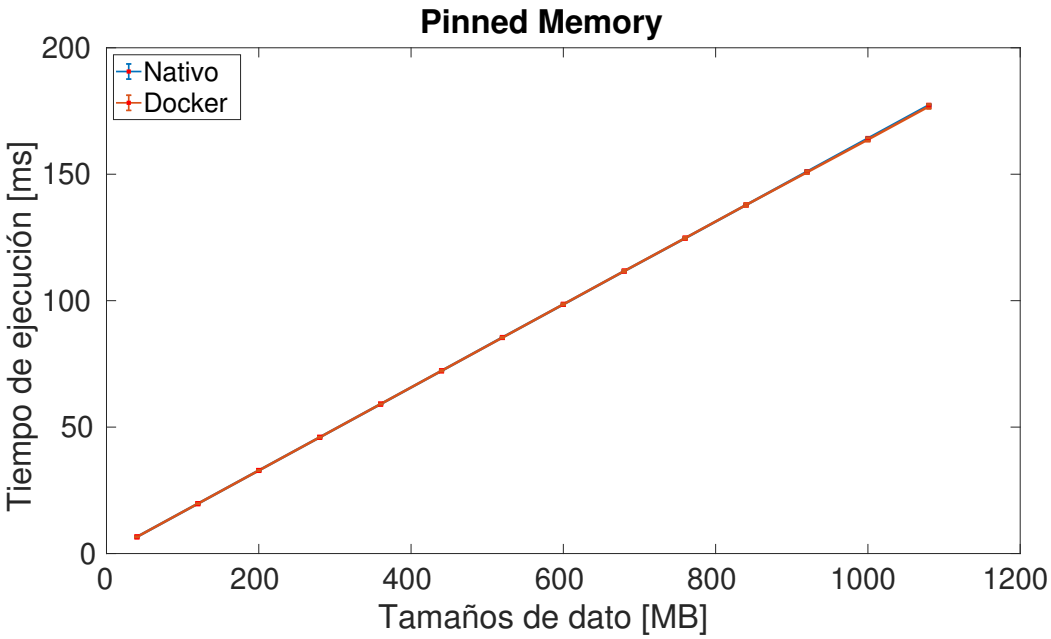
se almacenan en un tipo de memoria llamada *pegeable*; luego los datos pasan a un espacio de memoria llamado *pinned* y finalmente se envían a la GPU. Esta optimización almacena los datos directamente en la memoria *pinned* evitando el retardo por la transferencia entre la memoria *pegeable* y la memoria *pinned*.

Figura 6. La optimización de Pinned Memory se puede ver gráficamente de la siguiente forma. En el lado izquierdo se muestra la forma por defecto del flujo de datos entre *Host* y *Device*. El lado derecho muestra una alternativa que logra reducir el tiempos, gracias a la eliminación de una transferencia en *Host* (Tomado de [12])



El principal inconveniente del uso de memoria *pinned* es que está limitada y sobrecargarla reduce la cantidad de memoria física disponible para el sistema operativo y para otros programas. Para la prueba se repitió la metodología de las pruebas de transferencias, con 60 repeticiones de la prueba en donde cada una varía los datos que se envían de memoria *pinned* a la GPU desde 40 MB hasta 1160 MB. Los resultados para esta optimización se muestran en la Figura 7

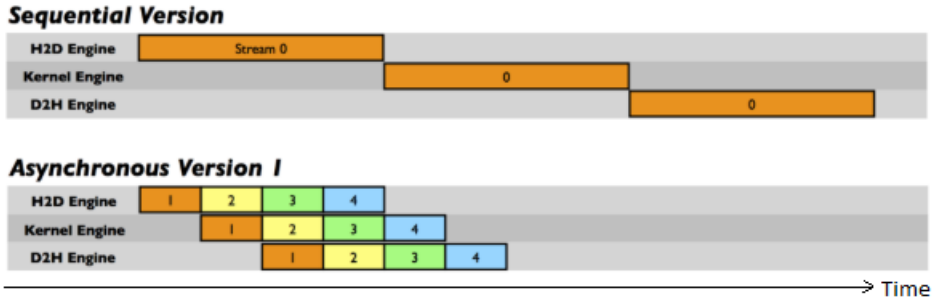
Figura 7. Resultados de tiempo empleado en una transferencia utilizando *pinned memory*, donde se varia el tamaño del dato que se transfiere.



4.1.5. *Overlapping* La segunda optimización que se realizó fue la de *overlapping*. Esta optimización permite analizar el uso de múltiples *streams* y desincronizar las operaciones. Los *streams* se pueden entender como conjuntos de instrucciones secuenciales. Lo que se define como un *stream* es entonces la secuencia de copiar un dato de CPU a GPU luego operarlo en GPU y luego retornarlo a CPU. El *overlapping* consiste en dividir el *stream* en varios *streams* que realizan la misma secuencia pero en diferentes partes del dato. Y mediante la desincronización (es decir que ahora no es necesario que un *stream* termine para que el otro comience) es posible solapar *streams* y disminuir el tiempo de ejecución. La Figura 8 muestra una comparación de una ejecución secuencial de tres etapas (Los datos de CPU a GPU, luego se operan en un kernel, y finalmente de GPU a CPU) contra una que hace *overlapping*. En ella se puede observar que el tiempo de ejecución se reduce

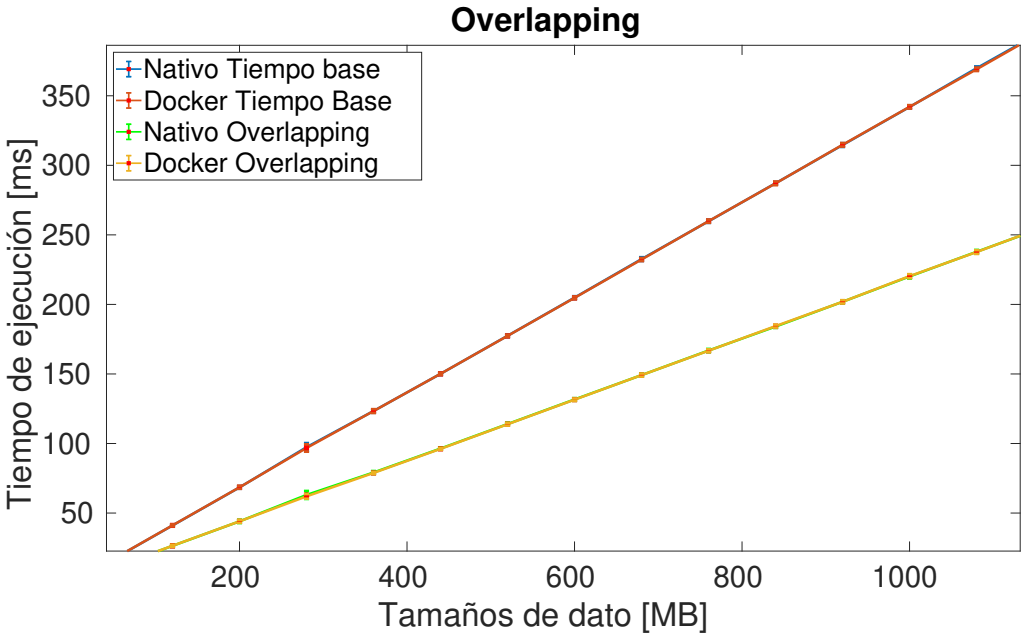
gracias a su asincronía y a la división en varios *streams*.

Figura 8. Comparación temporal de la implementación de un código de forma secuencial sin utilizar ninguna optimización y del mismo código usando *Overlapping* para disminuir el tiempo total de ejecución. (Tomado de [13])



Las pruebas sobre esta optimización se hicieron siguiendo la metodología anteriormente descrita en donde se hacen 60 repeticiones de cada intento con la variación del tamaño de los datos. En este caso se corrió una prueba sin las características de *overlapping* para tener una línea de base y comprobar que existiera una optimización. Los resultados de ambas pruebas (con y sin optimización) se muestran en la Figura 9

Figura 9. Resultados de tiempo empleado en código optimizado utilizando *overlapping*, y de ese mismo código sin optimización. El tamaño de los datos que se están procesando aumenta progresivamente.



4.1.6. Prueba Producto Punto Esta prueba se realizó con el objetivo de observar que sucede con Docker en un problema no tan trivial como lo es el cálculo de producto punto. Lo que hace especial a este caso es que si la implementación del producto punto se quiere calcular en la GPU es necesario hacer uso de memorias compartidas, de sincronización de hilos dentro de la GPU, y en nuestro caso incluimos una operación atómica. Específicamente se utiliza una suma atómica que realiza la operación de forma secuencial, pero conservando la integridad de los datos que suele estar comprometida cuando se realizan operaciones secuenciales de datos provenientes de operaciones en paralelo. Los resultados de esta prueba en Docker y en el sistema Nativo se encuentran en la tabla 4.

Tabla 4. Tiempos de ejecución y compilación de la prueba producto punto en GPU, utilizando el comando *time*. En la ultima columna se muestra la diferencia porcentual entre los resultados obtenidos de forma nativa y con Docker.

Producto Punto	**Docker [ms]**	**Nativo [ms]**	**Diferencia**
Compilación	0.5993±0.2627	0.5108±0.0123	**-17.3** %
Ejecución	0.9729±0.0165	0.6220±0.0114	**-56.4** %

Para comparar también se realizó la prueba del cálculo secuencial no en la GPU si no la CPU, lo cual implica una transacción más grande de GPU a CPU y una suma secuencial en CPU. Los resultados se presentan en la Tabla 5

Tabla 5. Tiempos de ejecución y compilación de la prueba producto punto en CPU, utilizando el comando *time*. En la ultima columna se muestra la diferencia porcentual entre los resultados obtenidos de forma nativa y con Docker.

Producto Punto Host	**Docker [ms]**	**Nativo [ms]**	**Diferencia**
Compilación	0.5459±0.0089	0.4965±0.0113	**-9.9** %
Ejecución	1.3159±0.0100	0.9648±0.0103	**-36.4** %

TRABAJO FUTURO

Como trabajos futuros se debería probar el rendimiento de Docker sobre no solo una GPU si no en *clusters* de varias GPUs, ya que suele ser común utilizar estos *clusters* en las investigaciones.

CONCLUSIONES

Es un punto clave en esta investigación, resaltar que los contenedores permiten el despliegue de aplicaciones de forma sencilla y son capaces de utilizar todos los recursos de hardware con los que se cuenta. Esto deja como única métrica de evaluación de contenedores de Linux el tiempo que estos pueden emplear para ejecutar la aplicación deseada.

Las pruebas demostraron que Docker ejecutando aplicaciones con transferencias de datos entre la GPU y la CPU no presenta perdidas de tiempo considerables, así mismo cuando se realizan optimizaciones como *Overlapping* o el uso de *Pinned Memory* tampoco se observa una gran diferencia en el tiempo de ejecución. Por otro lado en el caso del producto punto, se obtuvieron diferencias de hasta 56.4 $\%$, aun asi es razonable pensar que estas perdidas por la escala (milisegundos) llegan a ser muy difíciles de percibir.

Es necesario destacar el resultado obtenido debido a las al ciclo *for* (Figura 3), ya que fue la prueba en la que el rendimiento de Docker estuvo mas por debajo al obtenido en el sistema Nativo. Hay que resaltar que la prueba se realizó con iteraciones cercanas a los 300 millones de ciclos, que es un valor alto tomando como referencias trabajos consultados de aplicaciones de alto rendimiento ([7]). Pero otro aspecto de la prueba es que el contenido de este for, fue de escritura sobre punteros, y que en otros ciclos donde no se hacia escrituras sobre punteros no se reflejaban perdidas de tiempo. Lo que llevaría a concluir que lo que realmente afecta el rendimiento de Docker es la escritura sobre la RAM.

BIBLIOGRAFÍA

ARANGO, Carlos, Rémy DERNAT y John SANABRIA. “Performance Evaluation of Container-based Virtualization for High Performance Computing Environments”. En: *CoRR* abs/1709.10140 (2017). arXiv: 1709.10140. URL: http://arxiv.org/abs/1709.10140 (vid. pág. 17).

Chae, MINSU, HWAMIN Lee y KYEOL Lee. “A performance comparison of linux containers and virtual machines using Docker and KVM”. En: *Cluster Computing* 22.1 (2019), págs. 1765-1775. DOI: 10.1007/s10586-017-1511-2 (vid. pág. 18).

HARRIS, Mark. *How to Implement Performance Metrics in CUDA C/C++*. Ed. por Nvidia. https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/. 2012 (vid. pág. 23).

— *How to Optimize Data Transfers in CUDA C/C++*. Ed. por Nvidia. https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/. 2012 (vid. págs. 28, 29).

— *How to Overlap Data Transfers in CUDA C/C++*. Ed. por Nvidia. https://devblogs.nvidia.com/how-overlap-data-transfers-cuda-cc/. 2012 (vid. pág. 31).

Joy, A.M. “Performance comparison between Linux containers and virtual machines”. En: *Conference Proceeding - 2015 International Conference on Advances in Computer Engineering and Applications, ICACEA 2015* (2015), págs. 342-346. DOI: 10.1109/ICACEA.2015.7164727 (vid. pág. 17).

KURTZER, Gregory, Vanessa SOCHAT y Michael BAUER. “Singularity: Scientific containers for mobility of compute”. En: *PLOS ONE* 12.5 (mayo de 2017), págs. 1-20. DOI: 10.1371/journal.pone.0177459 (vid. pág. 17).

Linux. *Time Command Manual*. http://man7.org/linux/man-pages/man1/time.1.html (vid. pág. 23).

MANJARRÉS, Andrés. “Implementación De Un Módulo De Propagación De Onda Elástica 2D Utilizando Un Cluster De GPUS”. En: *Universidad Industrial De Santander* (2018) (vid. págs. 20, 35).

NVIDIA. *NVIDIA Visual Profiler*. https://developer.nvidia.com/nvidia-visual-profiler (vid. pág. 20).

SAHA, Pankaj y col. “Evaluation of Docker Containers for Scientific Workloads in the Cloud”. En: *CoRR* abs/1905.08415 (2019). arXiv: 1905.08415. URL: http://arxiv.org/abs/1905.08415 (vid. pág. 18).

Xavier, M.G. y col. “Performance evaluation of container-based virtualization for high performance computing environments”. En: *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013* (2013), págs. 233-240. DOI: 10.1109/PDP.2013.41 (vid. pág. 17).