Unidad Ejecutora:
Departamento de Ingeniería e Investigaciones Tecnológicas (DIIT).
UNLaM

Título del proyecto de investigación:
Optimización de la Recuperación de Documentos, usando como técnica base el LSI (Lematización Semántica Latente).

Programa de acreditación:
PROINCE

Director del proyecto:
Ryckeboer, Hugo Emilio Julio Ludovico

Co-Director del proyecto:
Spossito, Osvaldo Mario

Integrantes del equipo:
Gargano, Cecilia Victoria
Prilusky, Elisa Mirta
Barone, Miriam Andrea
Procopio, Gastón Emanuel
Quintana, Fabio Hernán

Fecha de inicio: 01/01/2015

Fecha de finalización: 31/12/2016

Informe final

Resumen:

Al evaluar los logros de las distintas etapas, que se realizaron durante la investigación precedente (2013-2014), surgieron ideas de mejora que, en bien de no demorar el objetivo de tener un prototipo, fueron recogidas, y no evaluadas durante el proyecto anterior pasando al presente.

Contando con un sistema funcionando, es sencillo sustituir componentes y ver si con ello se obtienen mejoras.

A continuación, se señalan los avances de las 3 observaciones heredadas, que fueron las inquietudes centrales de este proyecto.

1- La **Lematización** del idioma español disponible no daba resultados satisfactorios. En esta etapa se demuestra su mejora a realizar el procesamiento más rápido que la forma secuencial, se aceleró notablemente sin afectar la exhaustividad y relevancia.

2- Dada la posibilidad de extender la selección de documentos a corpus muy voluminosos, existen diversas ideas de subdividir el corpus en Grupos aplicando técnicas de Agrupamiento (**Clustering**), de modo tal de disminuir el espacio de búsqueda cuando se procesa una consulta. En este proyecto se incorporaron estas tecnologías a nuestro prototipo, con la intención de evaluar mejoras, pero se deberá reflexionar en un nuevo proyecto sobre la utilización conjunta de SVD (Descomposición en Valores Singulares) y agrupamiento.

3- Los sistemas que operan en gran escala deben recurrir necesariamente al uso en paralelo de varios procesadores. Se estudió la forma de **paralelizar** algunos algoritmos para acelerar adecuadamente los cómputos, demostrando que la mejora arroja rresultados muy positivos, los tiempos bajan drásticamente. Se logró distribuir en placas de video, pero esto abren nuevos interrogantes que son motivos de nuevos proyectos.

Palabras clave: Examinar, Indexar, Buscar, SRI, LSI (Browser, Indexing, Search, SRI, LSI)

Memoria Descriptiva

Según lo indicado en el protocolo de investigación inicial de la presente investigación, se ha cumplido en lo establecido en la programación de actividades que se informa en el Gantt, realizando las correcciones planteadas para esta segunda etapa y realizando las pruebas con las conclusiones del caso que se establecen en este informe.

Formación de recursos Humanos

- El grupo joven de desarrolladores fue capacitado en arquitectura en paralelo en placas de video. Gracias a este proyecto se formaron en esta tecnología

- CACIC XXI: Congreso Argentino de Ciencias de la Computación. El Congreso Argentino de Ciencias de la Computación (CACIC) es organizado por la Red de

Universidades Nacionales con carreras en Informática (RedUNCI). Se envió un trabajo. San Luis del 5 al 9 de Octubre de 2016

- CONAIISI 2016: Congreso Nacional de Ingeniería en Informática / Sistemas de Información. Universidad Católica de Salta UCASAL - Sede 2016: Campo Castañares - (Salta - Argentina) - jueves, 17 y viernes 18 de noviembre de 2016.

Introducción:

- **Selección del Tema**

Luego de concluido el trabajo realizado en la investigación: *Implementación de un Sistema de Recuperación de la Información*, al evaluar los logros de las distintas etapas, surgieron ideas de mejora que, en bien de no demorar el objetivo precedente de tener un prototipo, fueron recogidas, pero no evaluadas.

Contando con un sistema funcionando, es sencillo sustituir componentes y ver si con ello se obtienen mejoras.

A partir de esto, surge el presente trabajo

- **Definición del Problema**

Las técnicas de recuperación de textos que responden a inquietudes puntuales, se han hecho populares gracias a internet donde se ofrecen buscadores gratuitos. Estas técnicas no hacen referencia únicamente a grandes repositorios, sino que se desea incorporarlos en medianos y pequeños repertorios privados y para ello es necesario que existan buscadores hechos por profesionales de la computación. Los buscadores gratuitos no cubren las necesidades de los pequeños, quienes no desean poner al alcance de cualquier público sus contenidos. Además pueden contener expresiones no lingüísticas que requieren adaptar el análisis lexicográfico.

Al comenzar esta investigación, la propuesta de trabajo fue la optimización del Sistema de Recuperación de la Información (SRI) aplicando la Metodología LSI, construido precedentemente. El grupo se especializó en este método por las virtudes conocidas: asociación de sinónimos y desambiguación en términos de muchas acepciones (equívocos).

- **Justificación del Estudio**

Dada la posibilidad de extender la selección de documentos a corpus muy voluminosos la técnica básica se muestra lenta. Existen diversas ideas de subdividir el corpus en Grupos aplicando técnicas de Agrupamiento (Clustering), de modo tal de que al disminuir el espacio de búsqueda se acelere la consulta. En este proyecto se pretende dominar e incorporar esta tecnología a nuestro prototipo, con la intención de evaluar mejoras al mismo y la eficacia de esta tecnología.

Alternativamente los sistemas que operan en gran escala pueden recurrir al uso en paralelo de varios procesadores, también con la idea de acelerar la respuesta. Por ello se estudia la forma de paralelizar algunos algoritmos para acelerar adecuadamente los cómputos

- **Limitaciones**

El estudio se limita a un único modelo de placa y software de soporte ya que es prematuro poder establecer pautas válidas para cualquier placa.

Existen otras formas de paralelismo que no constituyen el objeto central de este proyecto que más bien se concentra en el uso de una placa de video como procesador auxiliar.

- **Alcances del Trabajo**

La idea principal del trabajo será emparejar por conceptos en lugar de por términos, es decir, un documento podrá ser recuperado, si éste comparte conceptos con otros que sean relevantes para una determinada consulta.

- **Objetivos**

Objetivos Principales:

- ✓ Lograr una mejor Lematización para Documentos en lengua española.
- ✓ Adquirir familiaridad en el uso de las técnicas de Agrupamiento.
- ✓ Acelerar algoritmos por uso de paralelismo.

Objetivos Secundarios:

- ✓ Mejorar el sistema de búsqueda ya construido

- **Hipótesis**

Que una mejora en la Lematización mejorará la calidad de la selección.

Que en repositorios voluminosos de documentos, se puede conseguir una mejora considerable en el tiempo de respuesta a las consultas, recurriendo tanto al Agrupamiento de los documentos como a la paralelización de los algoritmos utilizados.

Desarrollo

Para comprobar las mejoras, se realizaron distintas experiencias que dio énfasis a esta investigación, por lo tanto, a continuación, se describen las distintas ejecuciones que se realizaron en el proceso de tematización, la problemática de en aplicar clustering luego del SVD y finalmente se describe la utilización en paralelo de varios procesadores.

- **Material y Métodos**

1- Mejora en la velocidad de procesamiento del lematizador.

Introducción al problema.

Durante las distintas ejecuciones del proceso de lematización, se detectó que el tiempo de ejecución del mismo aumentaba considerablemente al incrementar el tamaño del corpus (cantidad de archivos que lo componen).

Procedimiento de la realización de la mejora.

Se comenzó realizando un seguimiento del código, para poder observar con más detenimiento cada instancia del proceso. En la inspección se detectó que, al terminar de obtener la raíz de una palabra, se debe buscar en un diccionario de lexemas para determinar si la raíz es correcta. Este procedimiento se realiza de forma secuencial, lo cual al analizar cada palabra hace que la búsqueda se siempre desde el inicio del diccionario.

La primera implementación de mejora fue poner un índice determinado para la primera letra de cada palabra, ya que el mismo está ordenado. Esto logró una reducción en el tiempo de ejecución porque la búsqueda se realiza a partir de la primera palabra de cada letra del abecedario, además se acorto la distancia de búsqueda.

Para la siguiente mejora se pensó en la utilización de procesamiento en paralelo y aprovechar el máximo rendimiento del procesador. Al momento de comenzar con la utilización de hilos, se debe mencionar que hubo muchas modificaciones para llegar a la versión actual.

En primer lugar, se tomó la porción de código que realiza la búsqueda de la raíz de la palabra, ya que de esta manera se realizan simultáneamente varias búsquedas con diferentes palabras. Una vez obtenida la raíz, se guardan los valores en una única lista que luego pasa a ser procesada con el procedimiento ya programado. De esta manera se logró una mejora considerable.

El siguiente paso fue lograr paralelizar el procedimiento en el cual se recorren los ítems para poder ubicarlos en las diferentes listas con las que cuenta el proceso. Hablamos de las palabras lematizadas, su cantidad de repeticiones dentro del corpus, en el documento y a qué documento corresponde. El primer obstáculo fue que las listas deben ser accedidas por un único proceso, ya que al agregarse nuevos ítems o modificar algún dato particular no puede ser realizado simultáneamente. Con lo cual este procedimiento no puede ser mejorado, sino todo lo contrario, se hace más lento debido a las esperas y bloqueos de subprocesos.

Se volvió a la versión anterior, y se revisó el procedimiento, lo cual llevó a una nueva lectura de esa porción de código, logrando así visualizar que la demora se debía a que la lista donde se guarda el cruce de la palabra y el documento junto a su repetición crecía rápidamente, lo que significaba que al buscar en ella (de forma secuencial) es lento y mayoritariamente se encontraban en la parte inferior de la lista.

Basados en el inconveniente mencionado, se decidió crear una lista temporal para procesar cada documento, lo que se logra, es disminuir el tiempo de búsqueda y agregar o modificar un ítem de una manera más rápida. Finalizada la lematización de ese documento, los resultados obtenidos, se suman a la lista con todos los resultados de los documentos ya procesados.

Introducción de ParallelFor y parallelForeach

Se ha introducido una nueva mejora, esta es la utilización del ParallelFor y ParallelForeach. La misma permite hacer paralización de procesos, en el caso del for paraleliza las iteraciones, mientras que en el foreach paraleliza por ítems (elementos). Se debe mencionar que la paralización se realiza a través del compilador, lo cual decidirá la cantidad de subprocesos a generar y de qué manera. Se quitaron las rutinas de creación de hilos por las de ParallelFor y ParallelForeach y se reutilizan las rutinas que los hilos utilizaban.

- **Resultados**

Las muestras obtenidas se visualizan en el ANEXO, al final del documento.

2- Problemática en aplicar clustering luego del SVD

Se realizó el procesamiento de indexación con 2000 artículos de diferentes portales de información, cada uno de estos artículos corresponde a diferentes temáticas (política, sociedad, policiales, etc.). La indexación corresponde a lematizar, calcularlos pesos locales, globales y normalizados, por último, aplicamos SVD a los datos normalizados. Se propuso utilizar clustering (con la metodología DBScan) con los datos provenientes del SVD y observar si se logra una mejora en la velocidad y relevancia en los documentos en la búsqueda. DBScan tiene una particularidad respecto de otros métodos de clustering, y es que siempre arma los mismos grupos (mientras no se alteren los paramentos de documentos mínimos que integran un grupo y una épsilon que establece la densidad del grupo) y además calcula cuantos grupos son posible generarse, mientras que K-means se debe indicar cuantos grupos se desea armar y los resultados varían en cada ejecución ya que dependen de los centroides elegidos aleatoriamente. Al momento de calcular la cantidad de grupos, se observó que solo armaba un único grupo, lo cual llevó al análisis del procedimiento. Cuando el algoritmo calcula las distancias entre los documentos, la diferencia que hay es a partir del décimo decimal, lo cual quedan todos en puntos resultantes son casi equidistantes y cumplen los paramentos de densidad del épsilon, dando como resultado que todos los documentos pertenezcan a un mismo grupo. Este fenómeno también pudo ser observado utilizando un corpus temático de 12.000 resoluciones jurídicas.

3- Uso en paralelo de varios procesadores.

Software y Equipos para la ejecución de las pruebas:

Primer equipo:

Sistema operativo: Windows 7.
IDE: Visual Studio 2013.
Placa de video: Nvidia G210.
Procesador: Intel I7-3770.
MotherBoard: Asrock H61M-VG3.
Memoria RAM: 8GB DDR 1.333 GHz

Segundo equipo:

Sistema operativo: Windows 8.1.

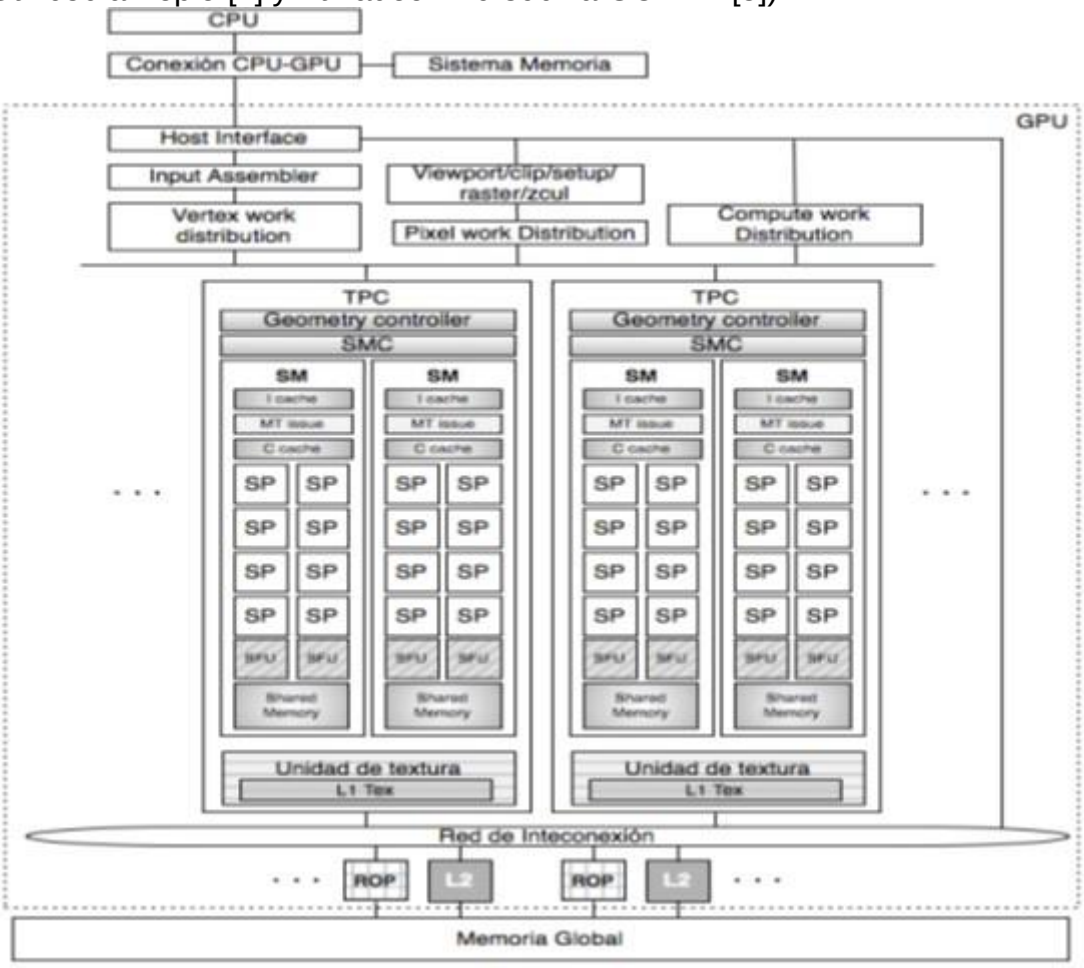
IDE: Visual Studio 2013.
Placa de video: Asus Geforce Gtx 650.
Procesador: Intel I7-3770.
MotherBoard: Asrock H61M-VG3.
Memoria RAM: 8GB DDR 1.333 GHz.

Tercer equipo:

Sistema operativo: Windows 8.1.
IDE: Visual Studio 2013.
Placa de video: Ati Radeon R9 390x.
Procesador: Intel I7-3770.
MotherBoard: Asrock H61M-VG3.
Memoria RAM: 8GB DDR 1.333 GHz.

Desarrollo de la investigación.

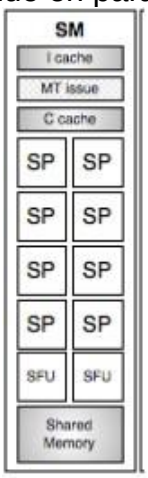
Se ha decidido comenzar con la investigación de la utilización de la GPU para poder realizar operaciones matemáticas y algebraicas de forma más rápida. Esto se debe a la disponibilidad de cálculos en paralelo que ofrece la GPU en relación a la CPU. Para ello se buscó información de diferentes bibliografías en las cuales se explica el procedimiento para programar en GPU y estudiar sus diferentes arquitecturas, ya que las placas de video utilizadas son de diferentes generaciones y empresa (Nvidia G210 pertenece a Tesla [3], Geforce Gtx 650 a Kepler[4] y Ati radeon R9 390x a GCN 1.1[5]).

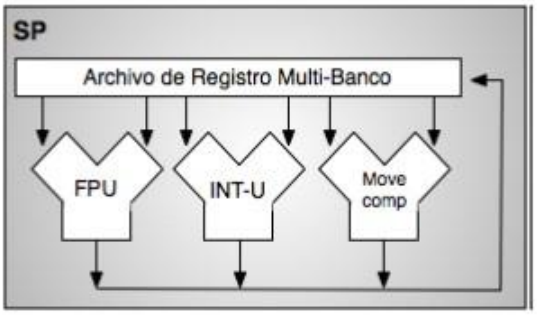


Arquitectura Nvidia

Las arquitecturas de Nvidia, Kepler y Tesla, tienen ciertos rasgos en común. En el gráfico se puede observar cómo esta´ compuesta una GPU y la conexión que hay con la CPU. En cuanto a la estructura interna, dependiendo del modelo de la placa se cuenta con una cantidad de TPC que almacena N cantidad de SM, y cada SM contienen M cantidad de SP y K cantidad de SFU.

Un SP es quién realiza una operación matemática y/o realiza un direccionamiento de datos en memoria. Las SFU contienen las funciones de punto flotante tal como la raíz cuadrada o funciones como seno, coseno, etc. El SM contiene una memoria compartida entre los diferentes SP, lo que permite el acceso a los datos de una manera más rápida a las diferentes unidades, sin la necesidad de tener que dirigirse a la memoria global. Cabe destacar que los datos pueden ser accedidos por cualquier SP perteneciendo al mismo SM, estos pueden modificar o simplemente leer el dato. Posteriormente se hablará de bloques y threads. Es conveniente mencionar que un SM representa un bloque, mientras que un SP es un thread. Un detalle a remarcar es que un núcleo de CPU, es más rápido y tiene una arquitectura más compleja que un SP, pero la diferencia radica en la cantidad de SP actuando en paralelo de lo que resulta que utilizando GPU, se supera a las CPU.

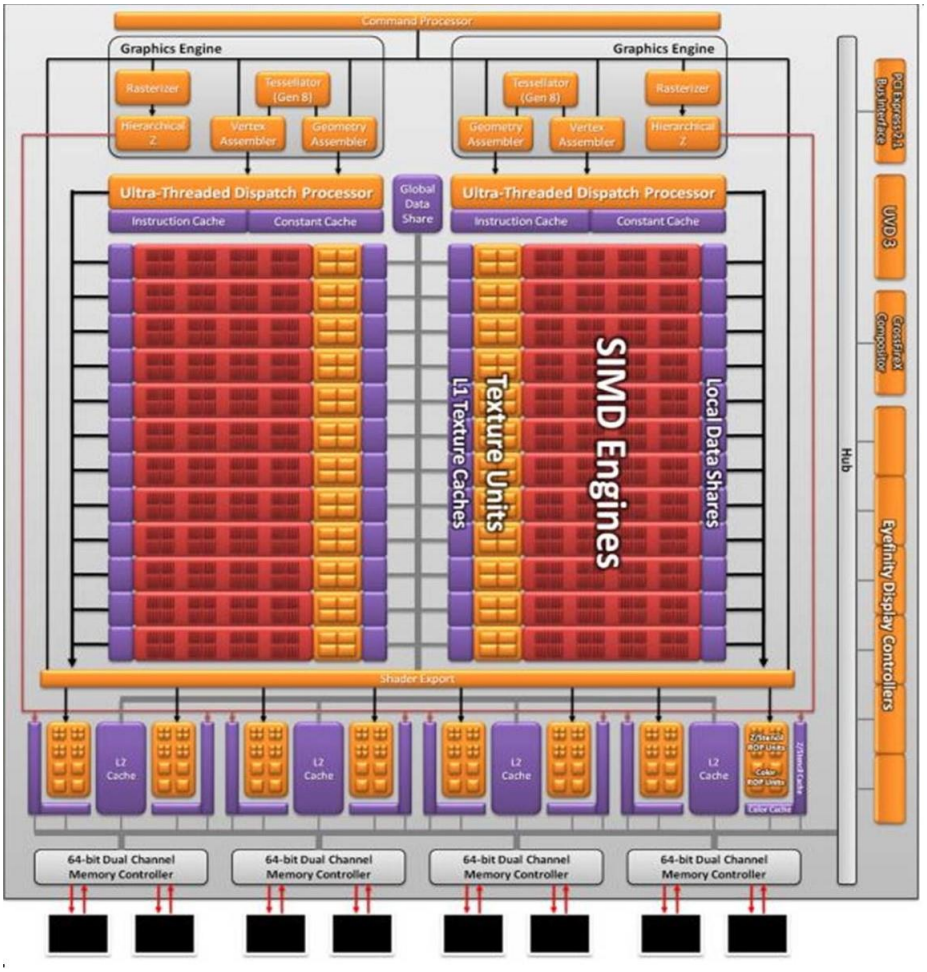




Los niveles de memoria que dispone, en un orden de mayor a menor en cuanto tamaño son: en primera instancia una memoria global, luego está la cache de nivel 2 (L2), seguido por la cache de nivel 1 (L1, que pertenece a cada TPC), la memoria compartida que dispone cada SM, y por último los registros dentro de cada SP. Un procedimiento estándar para la realización de un procesamiento de un dato conllevaría los siguientes pasos:

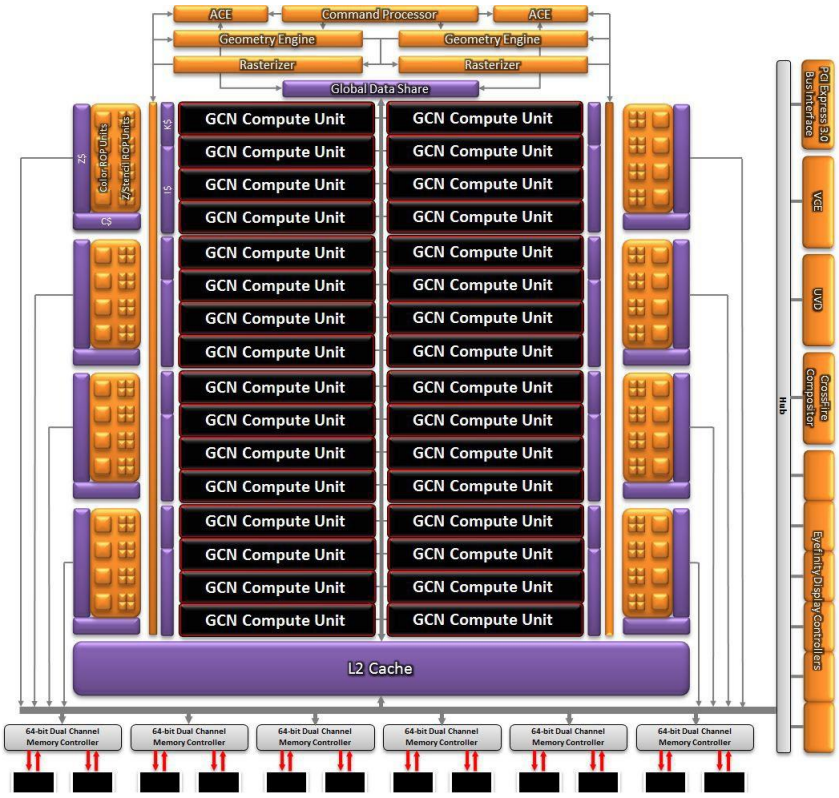
Primero se reserva una porción de memoria global (En la GPU), luego a través de la conexión entre GPU y la CPU (en nuestro caso PCI-Express) [2]. se envía los datos desde la RAM a la memoria global. Luego se envía la porción de código que será ejecutado en GPU, la misma configurará los diferentes TPC (de ser necesario 1 o más) para realizar la operación deseada. Los datos pasan los diferentes niveles de memoria mencionados anteriormente, con el mismo orden hasta llegar al SP que ejecutará las operaciones y guardará su resultado hasta realizar todo el procedimiento inverso de llegar a la memoria RAM. Una vez que el dato llegó a la memoria RAM se libera la memoria global reservada en la GPU.

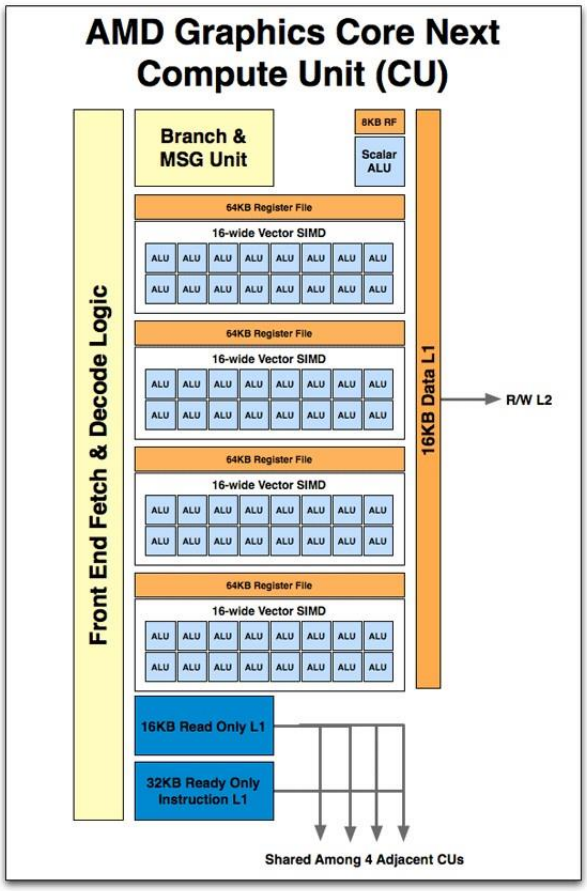
Arquitectura ATI.



En el caso de la placa de video de ATI, corresponde a la arquitectura GCN 1.1 podemos realizar una comparación en cuanto a la estructura de Nvidia. Si bien hay ciertos elementos que difieren, otros son iguales incluso en su funcionamiento.

En los que se pueden hacer una similitud en funcionamiento son: un TCP es el equivalente a SIMD Engines, un SM a CNG Compute unit y por ultimo un SP y a un ALU. Respecto a los niveles de memoria, poseen las mismas características, una memoria global, seguida de una cache de nivel 2 (L2) y nivel 1 (L1), finalizando con los registros de cada ALU.


ACE
Command Processor
ACE
Geometry Engine
Geometry Engine
Rasterizer
Rasterizer
Global Data Share
GCN Compute Unit
L2 Cache
Hub
PCI Express 3.0 Bus Interface
VCE
UVD
CrossFire Compositor
Eyefinity Display Controllers
64-bit Dual Channel Memory Controller


AMD Graphics Core Next
Compute Unit (CU)
Front End Fetch & Decode Logic
Branch & MSG Unit
8KB RF
Scalar ALU
64KB Register File
16-wide Vector SIMD
ALU
16KB Data L1
R/W L2
16KB Read Only L1
32KB Ready Only Instruction L1
Shared Among 4 Adjacent CUs

Proceso del desarrollo realizado

Una vez comprendido el dominio de la GPU, su funcionamiento y como es su arquitectura, se procedió´ a realizar diferentes programas con el objetivo de entender cómo utilizar y configurar su funcionamiento.

Para programar se ha utilizado la biblioteca “Cudafy”, del lenguaje C#, el cual nos permite utilizar “CUDA” [1] u “OPENCL”, “CUDA” es propio de Nvidia, mientras que “OPENCL” se puede utilizar para cualquier tipo de placa de video sin importar el fabricante. En base a lo expuesto, se ha decidido utilizar “OPENCL”. Adquirida la práctica y el entendimiento, se ha procedido a escribir el algoritmo propuesto por Ryckeboer Hugo en la publicación “Una paralización del método de Householder” [11].

Se subdividió el algoritmo en 5 etapas en las cual se puede aprovechar la paralización de las operaciones. Las operaciones que no requieren de paralización se realizan en CPU, mientras que el resto se ejecutan en GPU. También se trató de reducir las transferencias entre GPU y CPU para disminuir el tiempo de latencia y maximizar el tiempo de ejecución.

Debe aclararse ciertos aspectos fundamentales sobre la organización en la GPU, respecto a las grillas, bloques y threads. Las grillas y los bloques (en ATI GlobalWork, WorkGrup) poseen 3 dimensiones (X, Y y Z). Las grillas se componen de bloques, mientras que los bloques de threads. Cada bloque puede ejecutar una cantidad determinada de threads, en este caso podemos hablar de que un bloque en una dimensión puede ejecutar, por ejemplo, 512 threads. Entonces se podría configurar un bloque en (512, 0, 0) o (0, 512, 0), la dimensión Z tiende a ser menor a las dos dimensiones restantes ya que se utiliza en casos especiales en los que desee trabajar con 3 dimensiones. Otra una posible configuración es (22, 22, 0) en dos dimensiones, pero necesitaríamos uno o más bloques adicionales en caso que se tenga más de 484 threads. El mismo concepto de thread-bloque se aplica a grilla-bloque. Para poder aprovechar mejor la cantidad de threads en los bloques, se utilizó una sola dimensión, por consecuencia, una matriz es llevada a la forma de vector. De este modo para direccionar un thread a un espacio determinado del vector se utiliza el identificador del thread en el bloque + identificador del propio bloque * la cantidad de bloques, mientras que para una matriz se utiliza el identificador del thread en el bloque + identificador del propio bloque * la cantidad de bloques + la dimensión de una fila * I (I representa un número de fila).

A continuación, se explicará a grandes rasgos como se organizaron las operaciones del algoritmo. Se definirá los siguientes elementos:

S representa el valor comodín en el cual se puede elegir el signo, el mismo pertenece a la diagonal y es utilizado para llegar a la forma de Hessemberg.

F es un divisor de Vi para generar W.

Vi representa la reflexión de un vector U.

Wj es paralelo a U-V, tiene ceros donde esta diferencia es cero, los que se corresponden a los elementos que no sufren cambios.

K representa la columna en la cual se pretende generar 0.

Aij representa la matriz a diagonalizar.

La primera una parte se realza en CPU, donde se calcula los factores S y F. Y una parte en GPU en la cual se transcriben los valores de la fila a transformar de Aij al vector Vi, salvo en Vk donde se realiza la suma de S y la posición de la diagonal de Aij en la posición Akk . La segunda etapa se divide en dos procesos: En la etapa a. Se realizan los cálculos para construir el vector Wj calculando el producto V T * Aij fabricando WT. En la etapa b. Se realiza el producto escalar de F por el vector Vi en las posiciones de la columna que serán modificadas. La etapa 3 se realiza el producto escalar de F * Vk para evitar

enviar el vector V a la RAM para luego tener que realizar una carga nuevamente en la GPU, se utiliza un bloque con un único thread.

En la etapa 4 anula la columna K y se modifica la fila K realizando la operación de Aij = Vi * Wj. Estas etapas se repiten sucesivamente para cada K hasta realizar la diagonalización correspondiente a través del algoritmo expuesto.

Enseñanza.

- **Resultados**

La utilización de la GPU ha demostrado que tiene una gran capacidad para realizar operaciones matemáticas y algebraicas de formas más rápidas en comparación a la CPU, siempre y cuando, se estén realizando operaciones que puedan ser paralelizadas, en caso contrario se podría tener un costo de tiempo superior entre el envío de información entre la RAM y GPU, y viceversa. Se debe observar con detenimiento las operaciones realizadas, ya que, si se realiza modificaciones de información que este en el mismo lugar de memoria, los threads que acceden a ellas deben estar sincronizados. Por consecuencia de las ejecuciones de las diferentes pruebas entre las GPU Nvidia G210 y GTX 650, se observa que la GTX 650 tuvo una ventaja respecto al tiempo de ejecución de la G210. Además, se debe mencionar que la G210 no utiliza doble precisión para los decimales, con lo cual el error en los resultados es notable. Como consecuencia las diferencias entre las arquitecturas tienen un rol importante en la resolución del algoritmo.

Conclusiones

A lo que respecta a la lematización del idioma español disponible no daba resultados satisfactorios, se puede concluir que con el uso de los hilos se realiza un procesamiento más rápido que con la forma secuencial. A su vez el uso de las instrucciones parallelfor y parallelforeach ayuda a incrementar un poco más la velocidad de procesamiento.

En cuanto la posibilidad de extender la selección de documentos a corpus muy voluminosos, se deberá reflexionar sobre la utilización de SVD y clustering. Ya que el SVD proporciona una salida donde las diferencias de las distancias entre los documentos son muy cercanas, mientras que clustering utiliza esas distancias para poder agrupar los documentos.

Finalmente, respecto a los sistemas que operan en gran escala, estos deben recurrir necesariamente al uso en paralelo de varios procesadores. La utilización de GPU para realizar procesamiento de cálculos en paralelo, principalmente en matrices, tuvo un resultado positivo: los tiempos bajan drásticamente comparando un proceso secuencial en una Pc típica de escritorio contra el procesamiento sobre cualquiera de las GPU.

En cuanto a la comparación entre las diferentes GPU, se observa que los mejores tiempos se obtuvieron para la GPU R9 390X, a medida que haya más documentos, la diferencia de tiempos entre cada uno de ellas se va notando considerablemente.

Referencia y Bibliografía

Referencia

[1] CUDA Tool-kit *https://developer.nvidia.com/cuda-toolkit*

[2] PCI-Express *http://pcisig.com/specifications/pciexpress/resources*

https://nvlabs.github.io/moderngpu/performance.html

[3] Lindholm, Erik and Nickolls, John and Oberman, Stuart and Montrym, John,*NVIDIA Tesla: A unified graphics and computing architecture*,IEEE micro, 2008.

[4] NVIDIA Corporation,*NVIDIA Kepler GK110 Architecture Whitpaper*,2012.

[5] ATI,*AMD GRAPHICS CORES NEXT (GCN)ARCHITECTU-RE*,2012.

[6] Osvaldo Spositto, Gaston Procopio, and Fabio Quintana y´ Hugo Ryckeboer, *Una paralelización del método de Householder*,2016

Bibliografía

IEEE coma flotante http://chrishecker.com/images/f/fb/Gdmfp.pdf

Performance occupancy and latency

Matthew Scarpino,*OpenCL in Action: How to accelerate graphics and computation*, Primera edicio´n,2012.

Bhushan Rayrikar,*Parallel Implementation of the Singular Value Decomposition using OpenCL*,2011.

María Fabiana Piccoli, *Computación de alto desempeño en GPU*,2012.

Nagesh B. Lakshminarayana, Hyesoon Kim,*Effect of Instruction Fetch and Memory Scheduling on GPU Performance*

ANEXO

Pruebas de rendimiento.

Reporte que muestra el funcionamiento de la mejora realizada con respecto al código inicial y el actual.

Tiempo para 60 documentos

Tiempo del programa optimizado
Tiempo de indexación realizado con hilos 00:00:37.0115874
Tiempo sin optimizar
Tiempo de indexación realizado sin hilos 00:01:51.1555891

Tiempo para 150 documentos

Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:01:33.0030668
Tiempos sin optimizar
Tiempo de indexación realizado sin hilos 00:04:43.2907415

Tiempo para 250 documentos

Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:02:33.6472973
Tiempos sin optimizar
Tiempo de indexación realizado sin hilos 00:09:23.4354719

Tiempo para 350 documentos

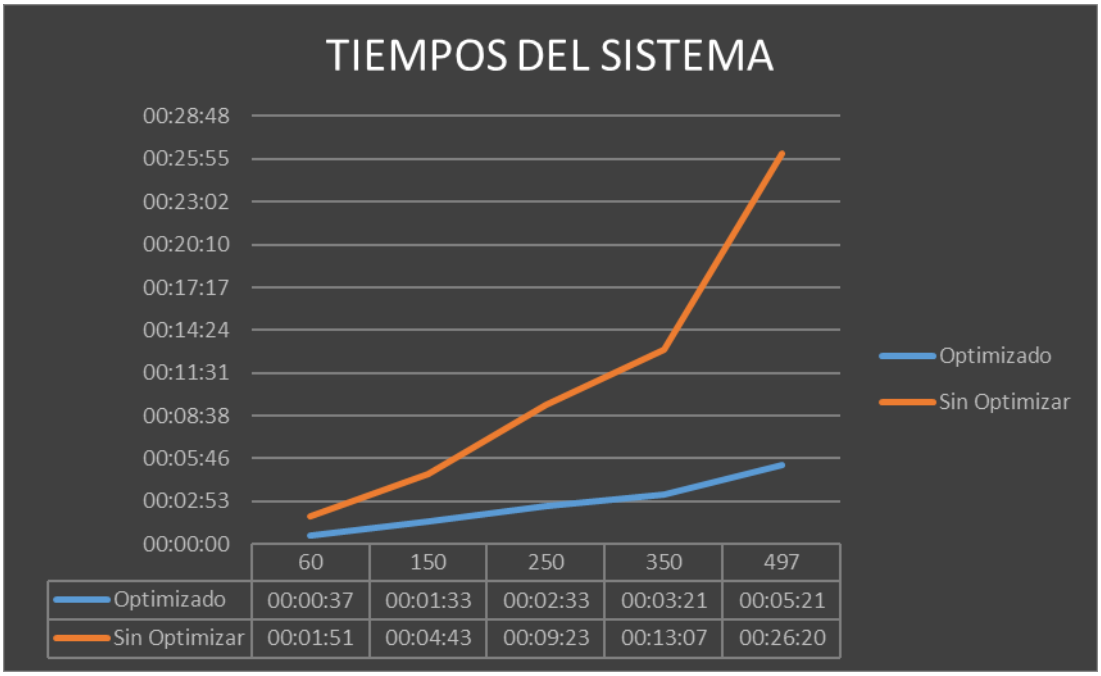
Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:03:21.1375793
Tiempos sin optimizar
Tiempo de indexación realizado sin hilos 00:13:07.3232358

Tiempo para 497 documentos

Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:05:21.0487581
Tiempos sin optimizar
Tiempo de indexación realizado sin hilos 00:26:19.5939878

Introducción de ParallelFor y parallelForeach

Gráfico de tiempos de ejecución.



	60	150	250	350	497
Optimizado	00:00:37	00:01:33	00:02:33	00:03:21	00:05:21
Sin Optimizar	00:01:51	00:04:43	00:09:23	00:13:07	00:26:20

Pruebas de rendimiento de hilos vs ParallelFor y ParallelForeach

Solo se prueba el algoritmo de lematización.

Tiempo para 150 documentos

Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:00:28.0010385
Tiempos sin optimizar
Tiempo de indexación realizado con ParallelFor y ParallelForeach 00:00:16.9447750

Tiempo para 250 documentos

Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:00:28.5490385
Tiempos sin optimizar
Tiempo de indexación realizado sin hilos 00:00:17.0801163

Tiempo para 350 documentos

Tiempos del programa optimizado
Tiempo de indexación realizado con hilos 00:00:29.0631051
Tiempos sin optimizar
Tiempo de indexación realizado con ParallelFor y ParallelForeach 00:00:16.9323335

Tiempo para 497 documentos

Tiempos del programa optimizado

Tiempo de indexación realizado con hilos 0:00:29.5276663
Tiempos sin optimizar
Tiempo de indexación realizado con ParallelFor y ParallelForeach 00:00:17.1816342

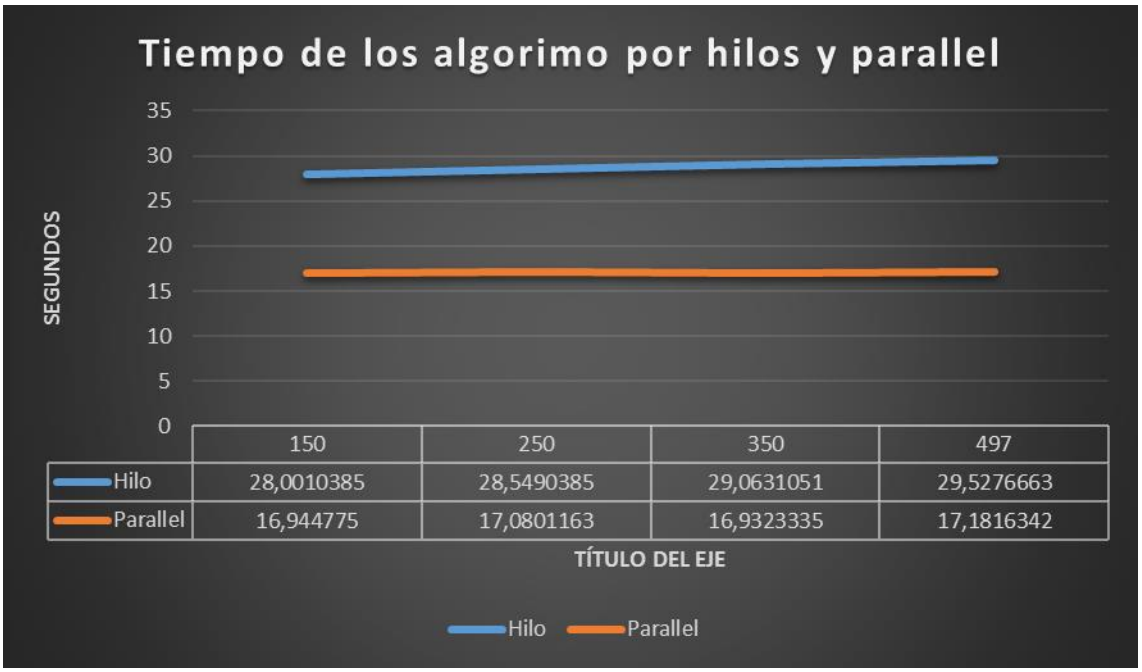
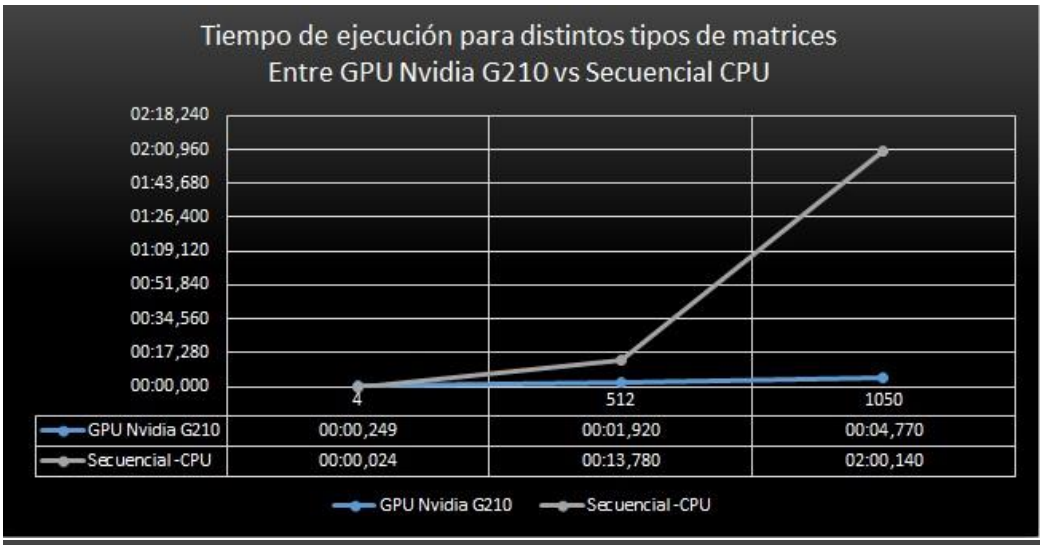
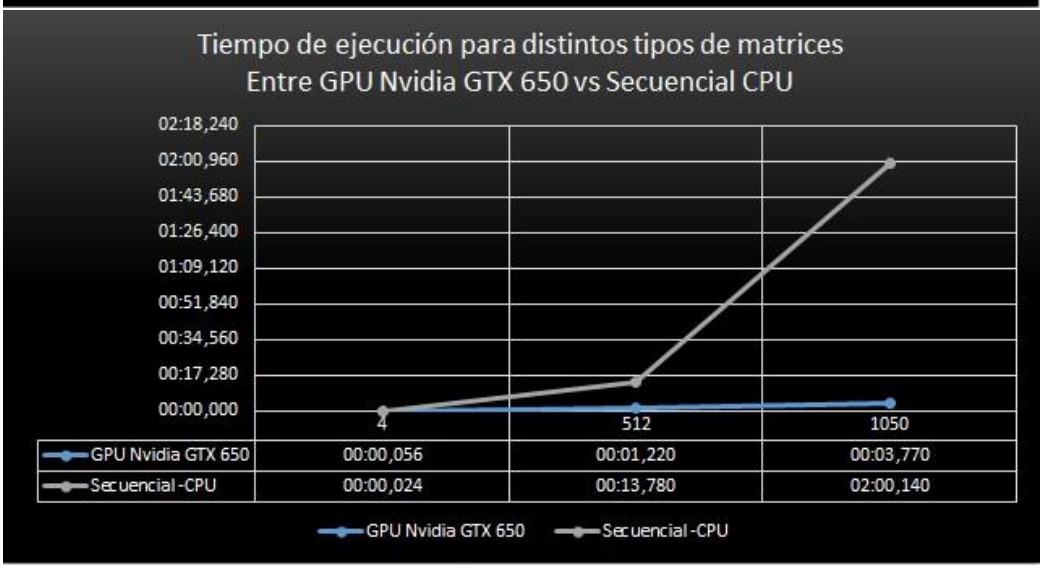
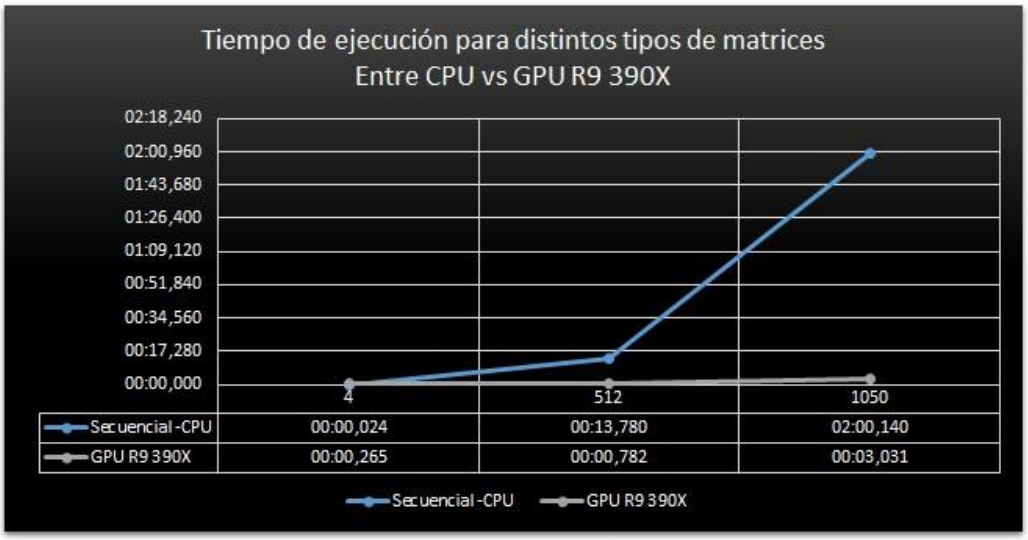


Gráfico de tiempos de ejecución.

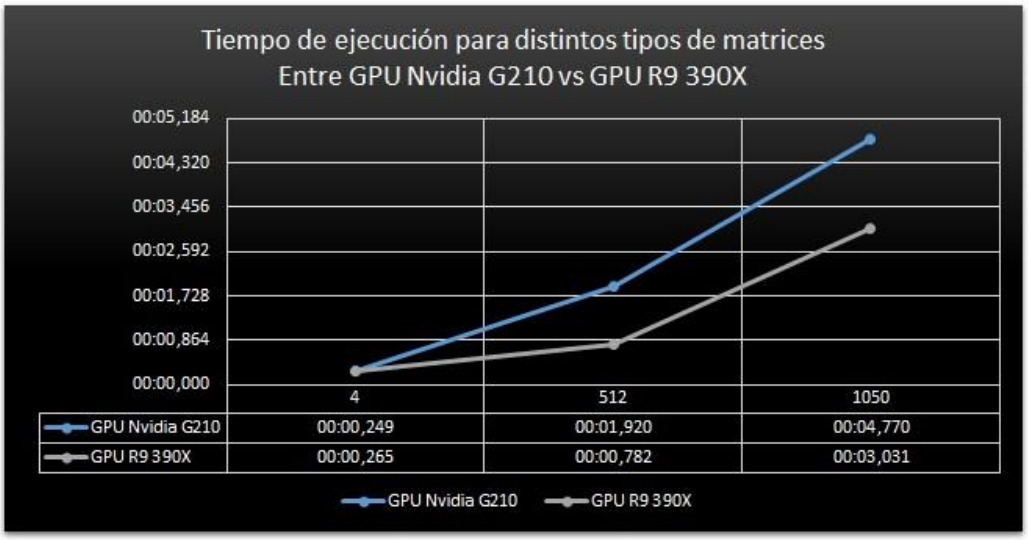

Tiempo de ejecución para distintos tipos de matrices
Entre GPU Nvidia G210 vs Secuencial CPU
02:18,240
02:00,960
01:43,680
01:26,400
01:09,120
00:51,840
00:34,560
00:17,280
00:00,000
4
512
1050
GPU Nvidia G210 00:00,249 00:01,920 00:04,770
Secuencial -CPU 00:00,024 00:13,780 02:00,140
GPU Nvidia G210
Secuencial -CPU

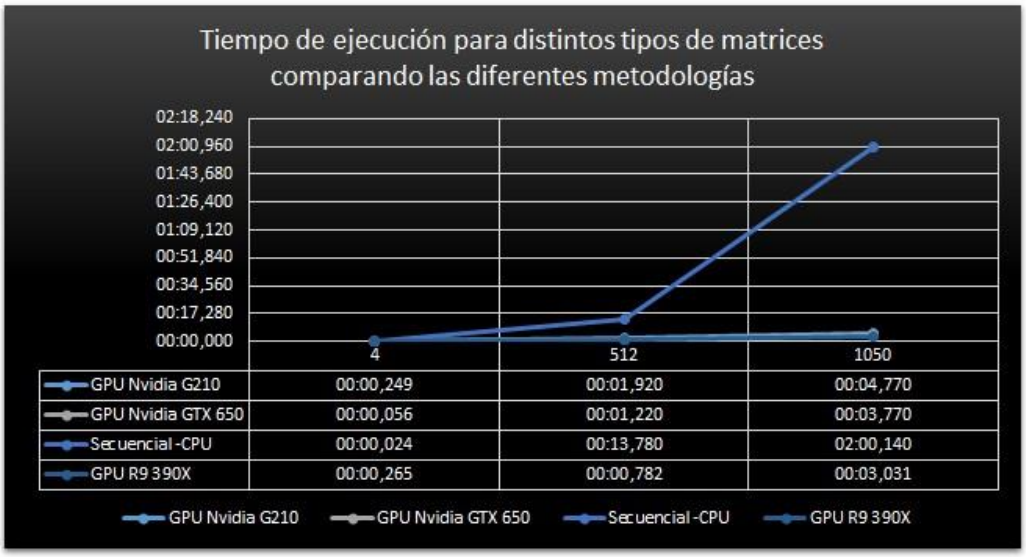

Tiempo de ejecución para distintos tipos de matrices
Entre GPU Nvidia GTX 650 vs Secuencial CPU
02:18,240
02:00,960
01:43,680
01:26,400
01:09,120
00:51,840
00:34,560
00:17,280
00:00,000
4
512
1050
GPU Nvidia GTX 650 00:00,056 00:01,220 00:03,770
Secuencial -CPU 00:00,024 00:13,780 02:00,140
GPU Nvidia GTX 650
Secuencial -CPU

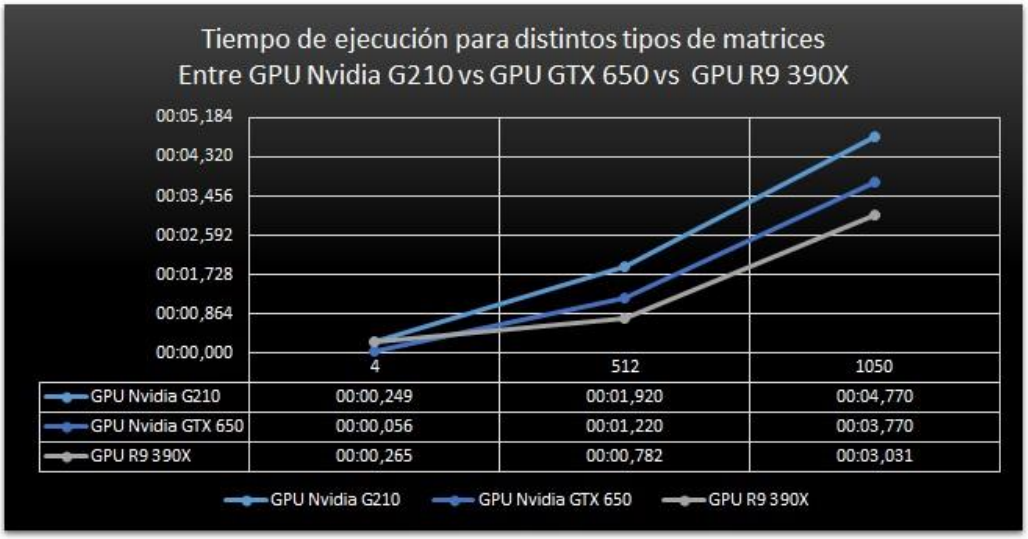

Tiempo de ejecución para distintos tipos de matrices
Entre la GPU Nvidia G210 y Nvidia GTX 650
00:05,184
00:04,320
00:03,456
00:02,592
00:01,728
00:00,864
00:00,000
4
512
1050
GPU Nvidia G210
00:00,249
00:01,920
00:04,770
GPU Nvidia GTX 650
00:00,056
00:01,220
00:03,770
GPU Nvidia G210
GPU Nvidia GTX 650


Tiempo de ejecución para distintos tipos de matrices
Entre CPU vs GPU R9 390X
02:18,240
02:00,960
01:43,680
01:26,400
01:09,120
00:51,840
00:34,560
00:17,280
00:00,000
4
512
1050
Secuencial -CPU
00:00,024
00:13,780
02:00,140
GPU R9 390X
00:00,265
00:00,782
00:03,031
Secuencial -CPU
GPU R9 390X


Tiempo de ejecución para distintos tipos de matrices
Entre la GPU Nvidia G210 y Nvidia GTX 650
00:05,184
00:04,320
00:03,456
00:02,592
00:01,728
00:00,864
00:00,000
4
512
1050
GPU Nvidia G210
00:00,249
00:01,920
00:04,770
GPU Nvidia GTX 650
00:00,056
00:01,220
00:03,770
GPU Nvidia G210
GPU Nvidia GTX 650


Tiempo de ejecución para distintos tipos de matrices
Entre GPU Nvidia G210 vs GPU R9 390X
00:05,184
00:04,320
00:03,456
00:02,592
00:01,728
00:00,864
00:00,000
4
512
1050
GPU Nvidia G210
00:00,249
00:01,920
00:04,770
GPU R9 390X
00:00,265
00:00,782
00:03,031
GPU Nvidia G210
GPU R9 390X


Tiempo de ejecución para distintos tipos de matrices
comparando las diferentes metodologías
02:18,240
02:00,960
01:43,680
01:26,400
01:09,120
00:51,840
00:34,560
00:17,280
00:00,000
4
512
1050
GPU Nvidia G210 00:00,249 00:01,920 00:04,770
GPU Nvidia GTX 650 00:00,056 00:01,220 00:03,770
Secuencial -CPU 00:00,024 00:13,780 02:00,140
GPU R9 390X 00:00,265 00:00,782 00:03,031
GPU Nvidia G210
GPU Nvidia GTX 650
Secuencial -CPU
GPU R9 390X


Tiempo de ejecución para distintos tipos de matrices
Entre GPU Nvidia G210 vs GPU GTX 650 vs GPU R9 390X
00:05,184
00:04,320
00:03,456
00:02,592
00:01,728
00:00,864
00:00,000
4
512
1050
GPU Nvidia G210 00:00,249 00:01,920 00:04,770
GPU Nvidia GTX 650 00:00,056 00:01,220 00:03,770
GPU R9 390X 00:00,265 00:00,782 00:03,031
GPU Nvidia G210
GPU Nvidia GTX 650
GPU R9 390X