




# Generación del frente de onda en paralelo mediante GPUs, con aplicación en tomografía sísmica

## Wavefront generation in parallel through GPUs, with application in seismic tomography

Alejandro Jimenez Xelhuantzi<sup>1</sup>, Alfredo Oscar Matlalcuatzi Sandoval<sup>2</sup>, José Federico Ramírez Cruz<sup>3</sup> ,  
José Crispín Hernández Hernández<sup>4</sup> , Edmundo Bonilla Huerta<sup>5</sup>   
Instituto Tecnológico de Apizaco, Carretera Apizaco - Tzompantepec esquina con Av. Instituto Tecnológico  
S/N, Conurbado Apizaco - Tzompantepec, Tlaxcala, México., C.P. 90300.<sup>1 2 3 4 5</sup>  
alexjx17@gmail.com<sup>1</sup>, oscarmatlalcuatzi@gmail.com<sup>2</sup>, federico\_ramirez@yahoo.com.mx<sup>3</sup>, josechh@  
yahoo.com<sup>4</sup>, edbonn@hotmail.com<sup>5</sup>

### PALABRAS CLAVE:

Calculo de diferencias finitas de  
tiempos de recorrido en paralelo,  
Cómputo paralelo, CUDA C

### RESUMEN

En el campo de la tomografía sísmica se realizan estudios del subsuelo que buscan comprender la composición de la corteza terrestre y localizar minerales, siendo esta última una de las prácticas más comunes. Para dichos estudios se implementan modelos de velocidades de ondas sísmicas a través de la corteza terrestre, causadas por fuentes naturales o artificiales. En este trabajo se presenta la paralelización de la generación del frente de onda implementada en GPUs de NVIDIA mediante CUDA. De esta manera se logra una aceleración de hasta 3.3 veces para 7 casos de prueba en comparación a la alternativa en cómputo secuencial, pero dicha aceleración puede ser mayor.

### KEYWORDS:

Parallel finite-difference calculation  
of traveltimes, Parallel computing,  
CUDA C

### ABSTRACT

In the field of the seismic tomography studies are made of the subsoil that seek to understand the composition of the earth's crust and find minerals, and the last one is one of the most common practices. For those studies seismic wave velocities models are implemented through the earth's crust, caused by natural or artificial sources. In this paper we present the parallelization of wavefront generation implemented in NVIDIA GPUs using CUDA. In this way we got an acceleration of up to 6.1 times is achieved for seven test cases in comparison to the alternative in sequential computing, but this acceleration can be greater.

Recibido: 4 de agosto del 2017    Aceptado: 21 de noviembre de 2017    Publicado: 15 de diciembre de 2017

## 1 INTRODUCCIÓN

En la Tomografía sísmica (TS) se estudia la corteza terrestre, principalmente en busca de minerales. En ella se realizan tareas computacionales robustas que demandan precisión en un tiempo de ejecución menor, para ello es necesario afrontar la problemática desde otros enfoques.

La TS es una técnica que permite generar una imagen que asimila observaciones del movimiento de la tierra recolectadas con sismómetros para mejorar los modelos estructurales y ha sido uno de los medios más efectivos para obtener imágenes del interior del planeta en las últimas décadas [1].

Los modelos de velocidades de ondas, propagadas a través de la corteza terrestre por fuentes de energía naturales, como terremotos; o artificiales, como explosiones, proporcionan información que puede ser utilizada para una amplia variedad de aplicaciones tales como investigaciones arqueológicas, control de calidad y evaluación de proyectos de ingeniería, además del descubrimiento de depósitos de agua, aceite o material de desecho [1, 2], por mencionar algunos.

En la investigación de Ramírez J. F. [3] se diseñó “Un algoritmo híbrido para el modelado de velocidades de la corteza terrestre”, el cual combina una estrategia evolutiva o algoritmos evolutivos (AEs), con el algoritmo de diferencias finita para el cálculo de tiempos de recorrido basados en el gradiente de J. E. Vidale [4, 5, 6].

El método basado en el gradiente necesita una buena aproximación inicial para hallar una solución óptima. Para resolver el problema inverso de encontrar un modelo de velocidades en tres dimensiones, la mayoría de los procedimientos necesitan un modelo de referencia inicial para desarrollar uno mejor.

Los AEs tienen la capacidad de buscar en espacios grandes y no son dependientes de una solución inicial aproximada a la óptima. Debido a la gran cantidad de parámetros que manipulan, los AEs se han utilizado principalmente en modelos bidimensionales. Además, han sido implementados con una combinación de diferentes técnicas para obtener mejores resultados [3]. Algunos de los inconvenientes son: el alto consumo de recursos computacionales y grandes tiempos de ejecución requeridos para completar algunas de las tareas; tal es el caso de

la generación de frente de onda, basado en cálculos de diferencia finita, y requiere de muchos procesos secuenciales.

En años recientes se ha buscado reducir los tiempos de ejecución y administrar mejor los recursos computacionales; haciendo uso de tecnologías que se han vuelto más accesibles y empleando nuevos enfoques; nos referimos específicamente al paradigma de cómputo paralelo, bajo la arquitectura de las tarjetas gráficas (Graphic Process Units GPUs) de NVIDIA mediante el lenguaje de programación CUDA. Un ejemplo de estas investigaciones es la de Ramírez J. F. y colaboradores que han propuesto un método híbrido, dividiendo la tarea global en tareas más específicas para generar modelos de ondas con mayor precisión en tiempos menores.

A continuación, se hace una breve descripción de la arquitectura de la GPU empleada en la presente investigación, así como el algoritmo utilizado para la generación del frente de onda. Se implementa el algoritmo de diferencias finitas de John E. Vidale [4] y es programado en el lenguaje CUDA para las GPUs de NVIDIA con un enfoque paralelo.

### Programación paralela con CUDA

CUDA C o simplemente CUDA es una plataforma de computación paralela y un modelo de programación que emplea una GPU para computación de propósito general NVIDIA [7]. CUDA puede describirse también como una colección de librerías en C para trabajar con las GPUs de NVIDIA.

Para entender el entorno CUDA es necesario tener claros algunos conceptos.

- Host (Anfitrión): hace referencia a la Unidad Central de Procesamiento (CPU).
- Device (Dispositivo): hace referencia a la Unidad de Procesamiento Gráfica (GPU).
- Kernel (Núcleo): código que se ejecuta de forma paralela en la GPU.
- Thread (Hilo): unidad mínima de ejecución en la GPU.
- Block (Bloque): agrupación de hilos, el número de hilos dentro de un bloque es limitado y está definido por

las características propias de la GPU a ser empleada.

- Grid (Cuadrícula): es a su vez un conjunto de bloques, definido en el anfitrión y enviado como parámetro a la GPU para ejecutar un núcleo, el tamaño de una cuadrícula es también limitado, pero permite que una gran cantidad de hilos sean ejecutados en una sola llamada a un núcleo.

El paradigma de programación paralela con CUDA es un híbrido entre los paradigmas secuencial y paralelo, ya que en una ejecución paralela se procesan simultáneamente sub tareas secuenciales que pueden ser independientes unas de otras y porque es una tarea secuencial global la que debe hacer el llamado y enviarle los parámetros necesarios a un núcleo para ser ejecutado en la GPU.

En otras palabras, esto quiere decir que del lado del anfitrión se ejecuta un programa “Principal” que realiza tareas secuenciales, prepara los datos que serán enviados a la GPU, define las dimensiones de la cuadrícula, envía los parámetros y el núcleo (código o tarea paralela) a ser ejecutado, en ese punto la GPU ha recibido la información necesaria para ejecutar el núcleo en cada uno de los hilos previamente definidos de la cuadrícula, una vez que el núcleo completa su ejecución se retornan los datos obtenidos por la GPU al anfitrión y este continua su ejecución secuencial, que pueden ser rutinas como: la presentación de los datos o su almacenamiento para su uso posterior por mencionar algunos ejemplos, después de esto se puede continuar ejecutando más núcleos o cualquier otra rutina deseada.

En la figura 1 se puede visualizar una representación global la distribución de los hilos en una GPU.

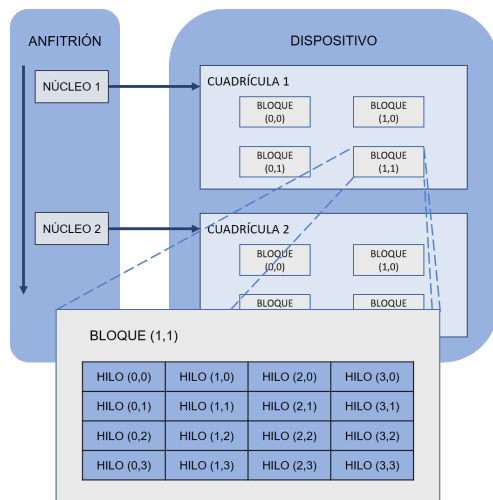


FIGURA 1. DISTRIBUCIÓN DE LOS HILOS.

### Sobre la memoria

Para comprender mejor el funcionamiento de una GPU es necesario saber cómo interactúan la memoria del anfitrión y de la GPU, algunas de las consideraciones a tener en cuenta son: la única comunicación que existe entre ellos es mediante la memoria del anfitrión y la memoria global de la GPU, no es posible acceder a ninguna otra memoria de la GPU a través del anfitrión, la información dentro de la GPU no es accesible en ejecución y la única manera de tener acceso a la información es después de la ejecución del núcleo y solo a la información contenida en la memoria global que es retornada al anfitrión, en la figura 2 se muestra una representación global de la memoria de una GPU.

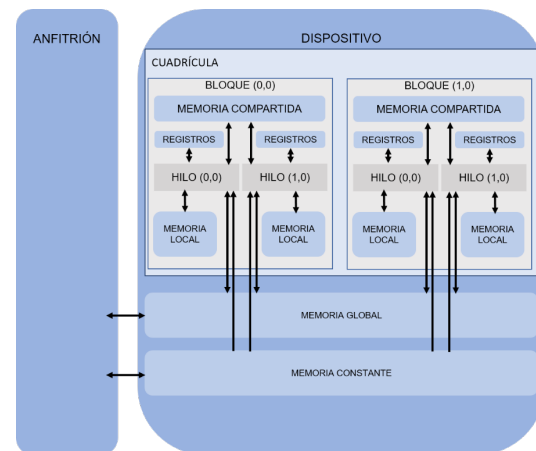


FIGURA 2. DISTRIBUCIÓN DE LA MEMORIA EN LA GPU.

El dispositivo cuenta con una memoria global la cual contiene datos modificables durante la ejecución del núcleo y una memoria constante que como su nombre lo indica no cambiara durante la ejecución, por lo tanto, el acceso a ella es más rápido que el de la memoria global, ambas pueden recibir datos directamente del anfitrión, pero, por otra parte, todos los demás tipos de memoria son exclusivos para el uso interno en la GPU.

Cada bloque cuenta con memoria compartida (esta memoria es de acceso más rápido que la memoria global) entre todos los hilos que pertenecen a ese bloque, pero inaccesible por otros bloques, cada hilo contenido en un bloque tiene a su vez una pequeña memoria local y registros propios que son inaccesibles por otros hilos, todos los hilos pueden tener acceso a la memoria global y constante, cuando múltiples hilos consultan un mismo espacio de memoria el acceso es secuencial y en el orden de petición.

Cabe resaltar que de acuerdo con los datos proporcionados por NVIDIA [7] el acceso a la memoria

compartida puede ser 10 veces más rápida que el acceso a la memoria global esto la convierte en una poderosa aliada para la aceleración de la(s) tarea(s) a paralelizar, pero es un recurso relativamente pequeño y limitado.

Implementación en paralelo del algoritmo de diferencias finitas

A continuación, se presenta la implementación en paralelo del algoritmo de diferencias finitas de J. E. Vidale [4, 5]. Las consideraciones iniciales a tener en cuenta son que existe una malla tridimensional uniforme la cual representa un prisma que es la sección del subsuelo a ser analizada, cada punto en la malla es identificado por una coordenada (x, y, z) “nodo”, se cuenta con una colección de datos iniciales para cada uno de los nodos en la malla, tales datos son las lentitudes (inversa de las velocidades), se debe considerar también la cantidad de fuentes sísmicas (shotpoints) a procesar, así como sus coordenadas de origen.

Para calcular el tiempo de recorrido de un nodo se consideran los nodos adyacentes a él y que corresponden a la capa de cobertura actual y de un tiempo anterior por lo cual se cuenta con 3 ecuaciones para 3 posibles esquemas propuestos por J. E. Vidale [4, 5], de ser posible calcular más de una ecuación entonces se registra como el tiempo de recorrido de ese nodo al valor menor de ellos, lo cual está representando al primer recorrido en llegar. A continuación, se presentan los algoritmos generales para la generación del frente de onda en paralelo y posteriormente se explican a mayor detalle.

**Algoritmo principal:**

Cargar los datos a procesar  
Calcular el tamaño de la cuadrícula  
Reservar el espacio de memoria para almacenar los resultados  
Ejecutar el programa núcleo (Parámetros de ejecución)  
Recibir los datos de la ejecución del núcleo  
Estructurar los datos  
Almacenar los datos

**ALGORITMO 1. ALGORITMO SECUENCIAL EJECUTADO POR EL ANFITRIÓN.**

Para la presente investigación se tiene el registro de las lentitudes almacenadas como un vector de tamaño conocido, entonces se procede a reservar espacio de memoria para los vectores donde se registran los tiempos de recorrido, es decir se generan tantos vectores de recorrido como fuentes sísmicas serán procesadas y de igual magnitud que el vector de lentitudes. Se recomienda emplear un vector para cada una de dichas colecciones de datos, dado que su manejo se simplifica en la GPU, pero esto no es obligatorio.

**Algoritmo núcleo:**

Para cada shotpoint  
Determinar el punto de origen en la malla  
Calcular los tiempos de recorrido para cada nodo en un cubo inicial de 5 x 5 x 5 unidades  
Mientras no se hayan alcanzado todas las fronteras del prisma  
Se expande la caja una unidad en cada dirección  
Se calculan los tiempos de recorrido para cada una de las caras de la caja  
Se calculan los tiempos de recorrido para cada una de las aristas de la caja, se procesa un nodo origen en la arista y después se pueden procesar paralelamente 2 caminos  
Se calculan los tiempos de recorrido para cada uno de los vértices

**ALGORITMO 2. ALGORITMO PARALELO EJECUTADO POR LA GPU.**

Para la implementación se inicia con un prisma uniforme al cual llamaremos “caja” que representa la cobertura “capa” a procesar en un tiempo, dicha caja se ira expandiendo uniformemente a sus 6 lados y dejara de ser uniforme conforme vaya alcanzando las fronteras del prisma que la contiene, las rutinas serán divididas de la manera siguiente: se procesan paralelamente cada uno de los 6 lados “caras” de la caja, después de haber procesado todas las caras se procesan paralelamente las 12 aristas de la caja, una vez completadas todas las aristas se procesan los 8 vértices (nodos) de la caja y se repite esta secuencia hasta que la caja alcance todas las fronteras del prisma, se ejecuta en paralelo este proceso para cada una de las fuentes sísmicas de las cuales se calculan sus frentes de onda.

Para la evaluación de esta investigación se han considerado 7 shotpoints para ser evaluados, por lo tanto, se emplean un máximo de 12 bloques por shotpoint para un caso máximo de 12 aristas. En cuanto a los hilos se refiere, se ha considerado que inicialmente en una arista se procesa un único nodo de inicio y después se pueden procesar paralelamente 2 caminos (2 nodos) y que por cada nodo se pueden evaluar paralelamente 3 ecuaciones, es por ello que el número máximo de hilos por bloque a emplear es de 6. Lo que nos da un total máximo de 72 hilos por shotpoint y finalmente para esta implementación se tiene un total máximo de 504 hilos trabajando paralelamente para calcular el frente de onda de los 7 shotpoints, es de esta manera como hemos definido las dimensiones de nuestra cuadrícula para ejecutar el programa núcleo.

Se inicia entonces el procesamiento del lado de la GPU y después de haber enviado todos los datos, así como de haber reservado el espacio de memoria para registrar los tiempos de recorrido del frente de onda, se da inicio a la tarea.

Lo primero a determinar es el punto origen en la malla, que es el punto más cercano a la coordenada origen de un shotpoint y que no precisa coincidir con un punto (nodo) de la malla, como se ejemplifica en la figura 3.

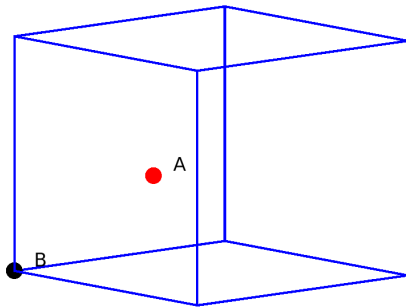


FIGURA 3. CUBO INICIAL PARA LA GENERACIÓN DEL FRENTE DE ONDA.

Donde el punto A representa la posición de origen del shotpoint y el punto B representa el nodo de la malla más cercano al punto A, a partir del punto B se determina un cubo inicial con un radio de 2 nodos de la malla, lo que resulta en un cubo de 5 x 5 x 5 nodos como se muestra en la figura 4.

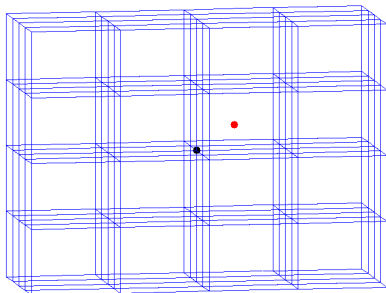


FIGURA 4. CUBO INICIAL DE 5X5X5.

Se calcula el tiempo de recorrido para cada uno de estos 125 nodos con la ecuación básica para calcular el tiempo: ecuación 1.

$$t=ds \quad (1)$$

Donde t es el tiempo a calcular, d es la distancia desde el punto A hacia un nodo y s es la lentitud correspondiente a ese nodo, los tiempos se registran en su vector de tiempos correspondiente en la coordenada equivalente. Esta tarea es llevada a cabo por un único hilo para cada shotpoint.

Una vez que se ha generado este cubo inicial de 5x5x5 se procede a calcular los tiempos de recorrido del perímetro correspondientes a un cubo de 7x7x7. Seis de los bloques de cada shotpoint serán empleados para procesar cada uno de los 6 lados que tiene la cobertura.

Mientras que ninguno de los lados de la cobertura haya alcanzado aún los límites de la malla, existirán 6 lados de la cobertura, si alguno de los lados ya ha alcanzado los límites de la malla global para ese lado, entonces el bloque correspondiente a ese lado estará en espera de que todos los demás bloques alcancen su límite de expansión, una vez que todos los bloques han alcanzado sus límites significa que se ha terminado la tarea para ese shotpoint.

Para cada lado de la cobertura se calculan los tiempos de recorrido de los vértices internos, como se ejemplifica en la figura 5, para el lado de una caja de 7x7x7, donde A representa el centro de la cara y los vértices internos son los que están encerrados en la cuadrícula.

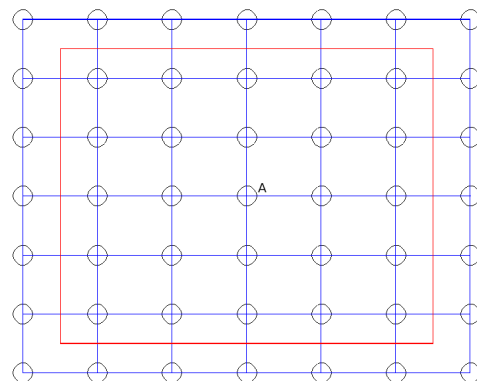


FIGURA 5. REPRESENTACIÓN DE UN LADO PARA UNA COBERTURA DE 7 X 7 X 7, DONDE LOS CÍRCULOS HUECOS REPRESENTAN LOS TIEMPOS DE RECORRIDO QUE AÚN NO HAN SIDO CALCULADOS Y EL CUADRO ENCIERRA A LOS TIEMPOS DE RECORRIDO INTERNOS DE ESE LADO.

Lo primero es determinar el orden en el que serán resueltos y esta tarea la lleva a cabo uno de los hilos del bloque que este procesando ese lado de la cobertura. Para ello se emplea un arreglo que contenga los 25 tiempos de recorrido de los nodos que están justamente detrás, en la cobertura anterior, se ordena el arreglo de menor a mayor y en ese orden son calculados los tiempos de recorrido para todos los nodos internos de ese lado de la cobertura. Después de este paso se sincronizan todos los hilos.

Para ejemplificar lo mencionado se asume que el nodo con el tiempo menor relativo a la cobertura anterior es el representado por el punto A en la cobertura actual de la figura 6, entonces la ecuación 4 para el esquema C es la única aplicable. En otros escenarios en los que ya se tengan calculados los tiempos de recorrido de nodos adyacentes en la cobertura actual al nodo A, entonces de ser posible se calcula la ecuación 3 para el esquema B y si es posible también se emplea la ecuación 2 correspondiente al esquema A, si más de una ecuación es aplicable entonces se registra el resultado menor como el tiempo de recorrido para ese nodo.

$$t_7 = t_0 + \frac{1}{\sqrt{2}} \sqrt{6h^2s^2 - (t_1 - t_2)^2 - (t_2 - t_4)^2 - (t_4 - t_1)^2 - (t_3 - t_5)^2 - (t_5 - t_6)^2 - (t_6 - t_3)^2} \quad (2)$$

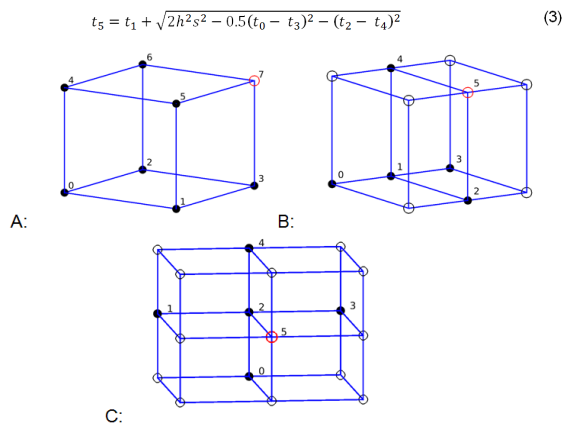


FIGURA 6. ESQUEMA DE LOS TRES CASOS BÁSICOS PARA EL CÁLCULO DE LOS TIEMPOS DE RECORRIDO PARA UN ESPACIO 3D UNIFORMEMENTE DISTRIBUIDO.

$$t_5 = t_2 + \sqrt{h^2s^2 - 0.25[(t_1 - t_3)^2 + (t_0 - t_4)^2]} \quad (4)$$

Para la ejecución de las ecuaciones 2 y 3 de los esquemas A y B respectivamente, se emplean 4 de los hilos disponibles del bloque, de ser posible cada hilo realizara el cálculo para cada uno de los 4 ángulos posibles en los que es posible aplicar los esquemas A y B, se registra aquel resultado con el valor menor. Al finalizar la ejecución de cada una de las 3 ecuaciones se sincronizan los hilos.

Una vez que se ha terminado de procesar las caras internas de cada lado de la caja se calculan los nodos internos de cada una de las 12 aristas figura 7, donde los círculos rellenos representan los nodos con tiempos de recorrido ya calculados y los círculos huecos dentro de los rectángulos rojos representan los nodos internos de las aristas a ser procesadas.

Para determinar en qué orden serán procesados los nodos de una arista se toman los tiempos de recorrido de la arista adyacente en un tiempo anterior y se almacenan en un arreglo, se ordena el arreglo de menor a mayor y en ese orden relativo se deben calcular los tiempos de recorrido para los nodos de la arista en el tiempo actual.

Se da el caso de que el primer nodo calculado no siempre está al extremo de la arista, de ser así en algunos casos es posible seguir 2 direcciones y calcular los tiempos de recorrido para 2 nodos paralelamente. Para calcular los tiempos de estos nodos es posible emplear las ecuaciones de los esquemas A y B, si ambas pueden ser empleadas se registra el resultado menor obtenido y se sincronizan los hilos al terminar su ejecución. Después de procesar todos los nodos internos de las aristas también se deben sincronizar los hilos.

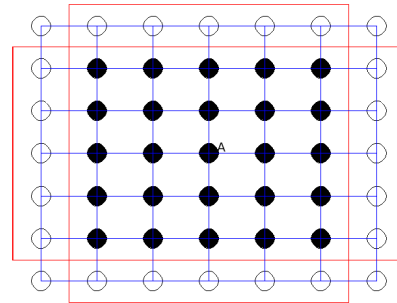


FIGURA 7. REPRESENTACIÓN DE UN LADO DE LA CAJA PARA UNA COBERTURA DE 7X7X7, DONDE LOS CÍRCULOS RELLENOS REPRESENTAN LOS NODOS DE LA CARA INTERNA QUE YA HAN SIDO PROCESADOS Y LOS CÍRCULOS HUECOS DENTRO DE LOS RECTÁNGULOS SON LOS NODOS INTERNOS DE LAS ARISTAS QUE AÚN NO SON PROCESADOS.

Finalmente, después de haber calculado los tiempos para todas las aristas se calculan los 8 vértices de la caja que corresponden a la cobertura actual, como se ejemplifica en la figura 8, donde los círculos rellenos representan los tiempos de recorrido ya calculados y los círculos huecos representan los nodos cuyos tiempos de recorrido deben ser calculados.



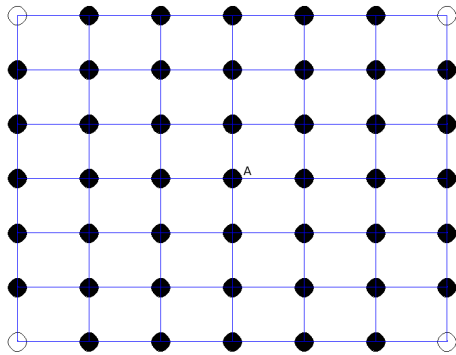


FIGURA 8. REPRESENTACIÓN DE UN LADO DE LA CAJA PARA UNA COBERTURA DE 7X7X7, DONDE LOS CÍRCULOS RELLENOS REPRESENTAN LOS NODOS O VÉRTICES DE LA CAJA DE LA COBERTURA ACTUAL.

Se continúa expandiendo el diámetro de la cobertura y se ejecuta este mismo proceso hasta que los 6 lados de la malla global hayan sido alcanzados y hasta que los tiempos de recorrido para todos los shotpoints hayan sido registrados.

Al haber terminado todas las operaciones correspondientes al GPU, los datos obtenidos son retornados al host para ser empleados como sea deseado.

Es importante recordar que después de cada tarea como la de ejecutar las ecuaciones, así como de terminar de calcular los tiempos de recorrido para las caras de la caja, aristas y vértices es necesario sincronizar los hilos para evitar cálculos erróneos.

Queda ahora por mencionar que la memoria compartida es un recurso poderoso tiene la limitante de ser escaso y dado que la generación del frente de onda es una tarea altamente secuencial, por ahora no nos ha sido posible desarrollar un método viable para fragmentar los datos en conjuntos más pequeños que puedan caber en el limitado espacio de la memoria compartida y es por ello que para todos los procesos descritos se ha empleado la memoria global, esto debido a la dependencia que tienen los nodos de una cobertura en tiempo  $t$  con todos los demás nodos adyacentes de la cobertura en un tiempo  $t-1$  pertenecientes al mismo lado de la caja.

#### Resultados y análisis

Para la evaluación de la presente investigación se emplea el volumen de datos de prueba correspondientes a una malla global de 231 km de ancho, 26 km de largo y 69 km de profundidad, discretizado en intervalos de 1km, que

producen una malla 3D con 414,414 nodos y se procesan 7 shotpoints correspondientes a 7 fuentes sísmicas artificiales controladas. Por lo tanto, se generan 7 frentes de onda, con 414,414 registros de tiempos de recorrido cada uno.

Se ha realizado la generación del frente de onda en forma secuencial 50 veces para cada uno de los 7 shotpoints, siendo un total de 350 ejecuciones con un tiempo promedio de ejecución de 0.2758 segundos.

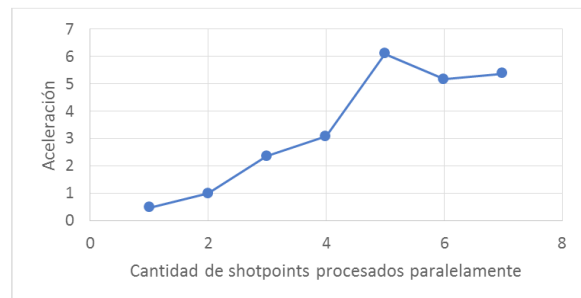
Para la versión en paralelo también se realizan 50 pruebas para procesar paralelamente de 1 a 7 shotpoints, los tiempos promedio para cada caso se pueden ver en la tabla 1. Es observable que a mayor cantidad de datos procesados se mejora la distribución de los recursos de la GPU e incluso con más datos se logran reducir los tiempos de ejecución.

Cantidad de shotpoints procesados paralelamente	Tiempo promedio de ejecución
1	0.5845
2	0.5492
3	0.3488
4	0.3568
5	0.226
6	0.320
7	0.367

TABLA 1. TIEMPOS DE EJECUCIÓN PARALELOS.

Los resultados de las pruebas secuenciales fueron comparados con los resultados de las ejecuciones en paralelo, cuyos valores resultaron ser iguales, por lo cual no hubo alteración en el cálculo.

A continuación, se presenta una gráfica de aceleraciones obtenidas de las ejecuciones paralelas con respecto a la alternativa secuencial.



GRAFICA 1. ACELERACIÓN OBTENIDA AL REALIZAR EN PARALELO LA GENERACIÓN DEL FRENTES DE ONDA

Es visible que al procesar 5 shotpoint paralelamente se logró obtener un pico de aceleración de hasta 6.1 veces más rápido que la alternativa secuencial y a pesar de haber sufrido una desaceleración al procesar 6 shotpoints, la aceleración comienza a crecer nuevamente y es evidente que a partir de del procesamiento paralelo de 2 shotpoint existe una aceleración aunque mínima con respecto a la alternativa secuencial y dichas aceleraciones van en crecimiento con respecto a un mayor número de frentes de onda a calcular.



### Conclusiones y trabajos futuros

Se puede concluir que la fragmentación de tareas robustas y altamente secuenciales permite dividir grandes tareas en otras tareas menores que pueden ejecutarse de manera paralela y lograr así mejorar los acelerar los tiempos de ejecución. En este ejemplo se logran buenas aceleraciones al procesar varios shotpoints y una visible aceleración creciente al procesar una mayor cantidad de shotpoints.

El principal reto a ser enfrentado al trabajar con paralelismo es la adecuada distribución de los recursos computacionales disponibles y completa responsabilidad de los desarrolladores el diseñar adecuadamente un pensamiento paralelo para la resolución de problemas, no hay una fórmula mágica que nos permita determinar la distribución optima de los recursos computacionales disponibles, es responsabilidad de la creatividad del o los desarrollares hacer una distribución adecuada para cada uno de los problemas a enfrentar. Siendo la administración de la limitada memoria compartida uno de los retos más interesantes en este enfoque de paralelismo, los limitados recursos físicos disponibles que llevan nos llevan a un punto de inflexión que puede llevarnos a la ejecución secuencial de un conjunto de procesos paralelos resultado en una ganancia nula del enfoque paralelo.

Los principales aportes han sido:

- Desarrollar una opción viable para generar el frente de onda de manera paralela.
- Reducir los tiempos de ejecución para el cálculo de tiempos de recorrido mediante diferencias finitas.

Como trabajo futuro nos hemos propuesto desarrollar una alternativa que nos permita fraccionar la cantidad necesaria de información para procesar las subrutinas actuales y esto a su vez nos permita hacer uso de los limitados recursos de la memoria compartida para lograr mayores aceleraciones, quedan algunas pequeñas tareas por paralelizar, algunas para las cuales sería necesario reservar mayores cantidades de hilos y memoria los cuales no serían utilizados después, pero el conjunto de todas estas mejoras en GPUs con los recursos disponibles pueden incrementar aún más la aceleración.

## REFERENCIAS

1. Rawlinson, N. Seismic tomography: A window into deep earth. *Physics of the Earth and Planetary Interiors*. 2010.
2. Backus, G., Gilbert, F. Constructing P-velocity models to fit restricted sets of traveltimes data. *Bulletin Seismological Society of America*. 1969, 59, 1407–1414.
3. Ramírez, J. F., Fuentes, O., Romero, R., Velasco. A Hybrid Algorithm for Crustal Velocity Modeling. *Advances in Computational Intelligence*. 2013.
4. Vidale, Jhon E. Finite-difference calculation of traveltimes in three dimensions. *GEOPHYSICS*. 1990, 55 (5), 521-526.
5. Vidale, Jhon E. Finite-Difference calculation of traveltimes. *Bulletin Seismological Society of America*. 1988, 78 (6), 2062-2076.
6. Vidale, Jhon E. Waveform effects of a high-velocity. *GEOPHYSICS*. 1987, 14, 542-545.
7. NVIDIA. CUDA C. Programming Guide. Recuperado el 3 de Marzo de 2017, <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>. 2017.
8. Blaise, B. Introduction to Parallel Computing. Lawrence Livermore National Laboratory 2016.
9. Eijkhout, V. Introduction to High Performance Scientific Computing. Public drafty. 2016. 2.
10. Carreón, E., Ramírez, J. F., Hernández, C. Algoritmo híbrido paralelo evolutivo para aplicaciones en tomografía sísmica. Instituto Tecnológico de Apizaco. 2015.
11. Cantú-Paz, E. A summary of research on parallel genetic algorithms. 1995.
12. Rüger, A. Aspects of Modern Raytracing Application Design, *Studia Geophysica et Geodaetica*. 2004, 48, 143-165.
13. Lewei, Mo., Jerry, M., Calculation of direct arrival traveltimes by the eikonal equation, *SEG Technical Program Expanded Abstracts*. 1994. 779-782.
14. Hole, J. A. Nonlinear high-resolution three-dimensional seismic travel time tomography. *Journal of Geophysical Research*. 1992.
15. Q. Liu, Y.J. Gu. Seismic imaging: From classical to adjoint tomography. Elsevier – Tectonophysics. 2012.
16. Dziewonski, A.M., Anderson, D.L. Preliminary reference Earth model. *Physics of the Earth and Planetary Interiors*. 1981, 25, 297-356.
17. Julian, B. R. and D., Gubbins. Three-dimensional seismic ray tracing. *J. Geophys.* 1977, 95-114.
18. Cerveny, V., I. A. Molotkov and I. Psencik. Ray methods in seismology. University of Karlova Press, Prague. 1977.
19. Kirk, D. B. and Hwu, W.-m. W. Programming Massively Parallel Processors: A Hands on Approach. Morgan Kaufmann Publishers Inc. 2010.1.

