Algoritmo evolutivo paralelo para aplicaciones en tomografía sísmica

Eustolia Carreón[1], José F. Ramírez[1], Miguel O. Arias[2], Edmundo Bonilla[1], Roberto Morales[1]

[1] Instituto Tecnológico de Apizaco, Apizaco, Tlaxcala, México

[2] Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, México

{euce_79,edbonn,moralescaporal}@hotmail.com, federico_ramirez@yahoo.com.mx, ariasmo@inaoep.mx

Resumen. En este trabajo se realiza la paralelización sobre una Unidad de Procesamiento de Gráficos de la función de evaluación de una Evolución Diferencial (ED) que tiene como objetivo generar un modelo inicial de velocidades sísmicas en un volumen de la corteza terrestre. La función de evaluación incluye un algoritmo que traza los rayos sísmicos generados a partir de 7 fuentes de energía (shotpoints) hacia cientos de receptores (geófonos). En la etapa de paralelización, el cálculo del tiempo residual y el trazo de la trayectoria del rayo sísmico es asignado a una unidad mínima de ejecución en la GPU (hilo), que para la aplicación realizada en este trabajo fueron 4,440 ejecutándose simultáneamente en paralelo. Al ejecutar el algoritmo secuencial, el tiempo estimado fue de 8 segundos aproximadamente, mientras que en la versión paralela fue de 0.1 segundo. Los datos usados en este trabajo se obtuvieron de un experimento realizado en el campo volcánico El Potrillo, ubicado en el sur de Nuevo México, a cargo del Departamento de Ciencias Computacionales y del Departamento de Ciencias Geológicas de la Universidad de Texas, en El Paso.

Palabras clave: evolución diferencial, algoritmo de cobertura de rayos, CUDA, cómputo paralelo, unidad de procesamiento de gráficos.

1. Introducción

La tomografía sísmica es una técnica de imagen que procesa 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]. Un estudio de tomografía sísmica permite obtener un modelo de velocidades de la estructura de la corteza terrestre bajo cierta región. El conocimiento de estas velocidades tanto en longitud, amplitud y pro-

fundidad del volumen permite la localización de diferentes elementos que se encuentran bajo la tierra, por ejemplo: tipos de rocas, fluidos, gases, material de desecho, etc., debido a que cada uno de ellos presenta diferente velocidad de transmisión de ondas sísmicas de acuerdo a su densidad [2]. Para ello se usan diversas técnicas de búsqueda, en este caso en particular se utilizan los Algoritmos Evolutivos (AE), ya que tienen la capacidad de buscar en espacios grandes y no son dependientes de una solución inicial aproximada a la óptima como los métodos tradicionales basados en el gradiente; sin embargo, conlleva grandes costos computacionales, ya que generalmente este tipo de algoritmos generan poblaciones de soluciones candidatas, repitiendo el proceso muchas veces [3] y por lo tanto ralentiza la ejecución por horas o días, dependiendo de la cantidad de los datos que se usen. Debido a que muchas aplicaciones actualmente requieren mayor poder de cómputo del que una computadora secuencial es capaz de ofrecer, el cómputo paralelo ofrece la distribución del trabajo entre diferentes unidades de procesamiento, resultando mayor poder de cómputo y rendimiento del que se puede obtener mediante un sistema tradicional de un solo procesador [4]. Con el desarrollo de herramientas de programación de Unidades de Procesamiento Gráfico (GPUs, por sus siglas en inglés), varios algoritmos han sido adaptados a este hardware satisfactoriamente y la plataforma de computación híbrida GPU - CPU, ha alcanzado aceleraciones importantes, comparada con las implementaciones sobre CPUs únicamente [5]. En los últimos años se han implementado los AE en GPUs y se ha visto que son capaces de mejorar decenas de veces el desempeño mediante el uso de este hardware, sobre todo cuando se utilizan tamaños grandes de población [6].
En este trabajo se paraleliza, sobre una GPU, la función de evaluación de una ED, para generar un modelo inicial de velocidades sísmicas en un volumen de la corteza terrestre.

En la sección 2 se mencionan los módulos de un Algoritmo de Tomografía Sísmica (ATS). En la sección 3 se explica la ED usada y en la sección 4 se muestra el modelo paralelo. En la sección 5 se presentan resultados experimentales obtenidos y se hace una comparación entre el desempeño de las versiones secuencial y paralela. En la sección 6 se mencionan las conclusiones y los trabajos futuros.

2. Generación de frente de onda y cobertura de rayos

Diversos procesos constituyen un ATS [2], de los cuales en esta investigación sólo se consideran: la simulación de un frente de onda, la cual genera, en función de un conjunto de velocidades iniciales, los tiempos de llegada de las ondas sísmicas a partir de las fuentes de energía, hacia cada uno de los vértices de un modelo discreto del volumen de la corteza terrestre en espacios de 1 km^3 (Fig. 1); y el proceso de cobertura de rayos sísmicos, el cual se encarga de hacer el trazo de los rayos sísmicos siguiendo su trayectoria de forma inversa; es decir, desde el origen de los geófonos (destino del frente de onda generado) hacia cada una de las fuentes de energía (origen del frente de onda generado); considerando únicamente los que llegan a las fuentes y excluyendo los que se salen del volumen de estudio.

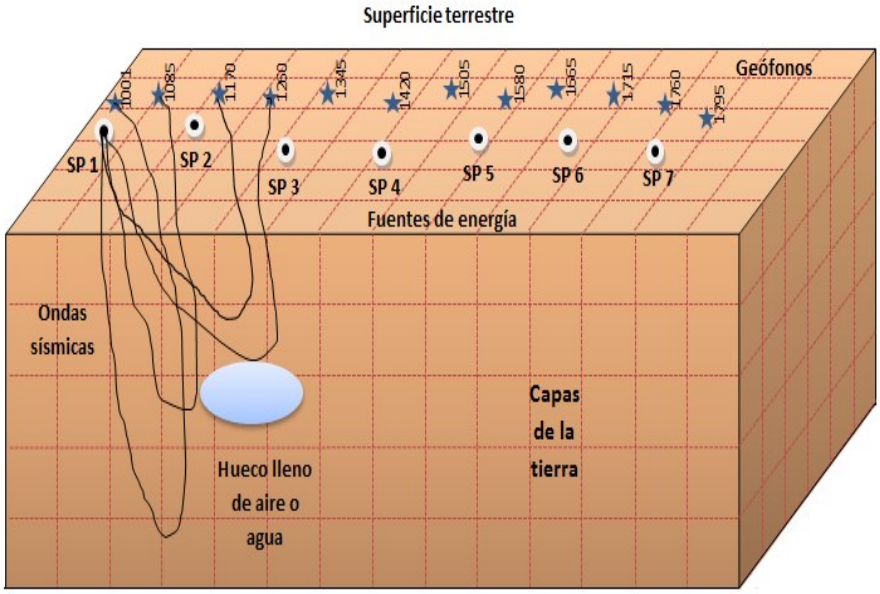

Fig. 1. Modelo discreto de la corteza terrestre

2.1 Representación de individuos

En el algoritmo, cada individuo de la población representa un conjunto de profundidades y velocidades en diferentes capas de la tierra, para este caso se tomaron 8 puntos de la profundidad debido a que anteriormente, con el uso de Estrategias Evolutivas, ha dado mejores resultados en comparación con otros valores [2]; sin embargo, este número puede variar de acuerdo a la elección de los usuarios. En la Fig. 3 se representa un individuo, el cual al ser visto como un vector, está dividido en dos partes. En la primera, los valores 1 y 69 ocupan las posiciones 0 y 7, respectivamente, correspondientes al mínimo y máximo de la profundidad en kms, y las 6 posiciones intermedias son datos de tipo entero, diferentes entre sí, y que deben ser ordenados de menor a mayor. En la segunda parte del individuo, se asignan las velocidades en un rango de 3 a 8 km/s; que cumplen con las mismas restricciones de la primera pero son valores reales. Las velocidades para las 61 capas restantes de las 69 en total se calculan mediante interpolación lineal.

La representación de cada individuo muestra que a cada valor de profundidad le corresponde una velocidad, debido a que la finalidad fue generar un modelo de capas planas, donde cualquier punto dentro de toda la superficie de una capa en determinado valor de profundidad tiene la misma velocidad; y aunque se sabe que en la realidad no es así, es usado de esta manera para obtener un modelo inicial de velocidades y realizar posteriormente una simulación de frente de onda.

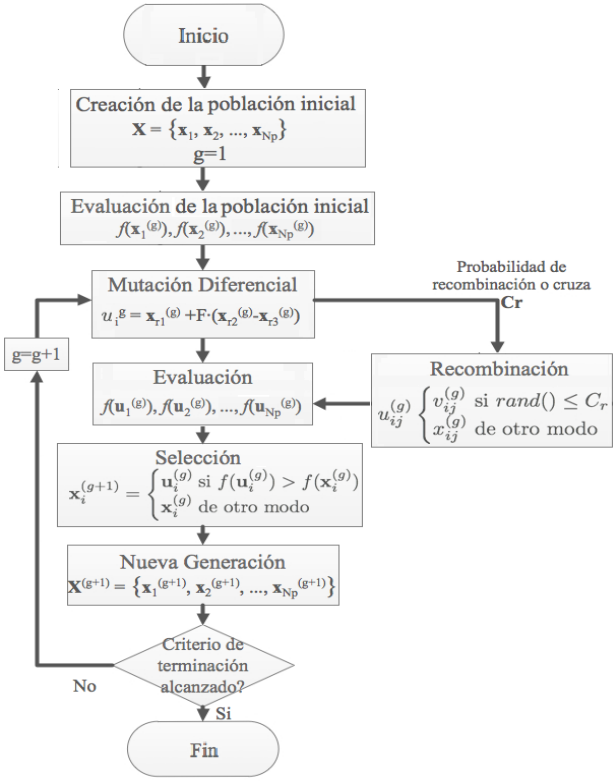

Fig. 2. Diagrama de flujo de una ED

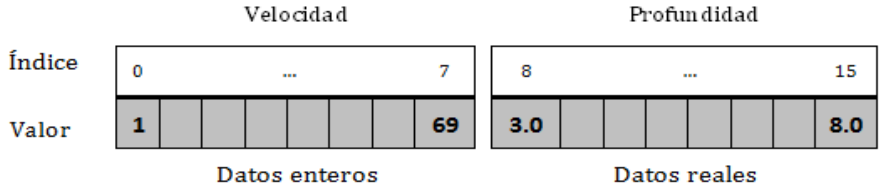

Fig. 3. Representación de un individuo

2.2 Mutación

El operador de mutación proporciona diversidad a la población a través de la introducción de nuevas soluciones, evitando la convergencia prematura en el algoritmo; es decir, la posibilidad de una rápida obtención no necesariamente del óptimo global. Para superar este problema es necesario conservar la diversidad en las generaciones, mediante un parámetro denominado factor de escala (F), elegido por el usuario para controlar la amplificación de la diferencia entre 2 individuos, así como para evitar el

estancamiento de la búsqueda, permitiendo la exploración de otras áreas en el espacio de búsqueda [8]. Su valor es tomado del rango de 0 a 1 [7]; de 0 a hasta 1.2 [10]; o bien es establecido a 0.5 [11]. Para este trabajo se asignó a F un valor aleatorio entre 0 y 1. La mutación se implementó mediante la siguiente ecuación

$$\vec{u_j} = \vec{x_{r1}} + F.(\vec{x_{r2}} - \vec{x_{r3}}) \tag{1}$$

Donde $\vec{u_j}$ es el vector mutante de la población y r1, r2, r3 son vectores de la población representados por valores enteros desde 0 hasta el número total de individuos, los cuales son diferentes entre sí y diferentes del índice en funcionamiento. En este trabajo el 80% del total de la población fue generado mediante este operador.

2.3 Recombinación o Cruza

La recombinación o cruza es una de las operaciones evolutivas implicadas en la generación de nuevos individuos. Esta operación es llevada a cabo con la participación de dos o más padres, quienes heredan rasgos o características a sus descendientes mediante una mezcla de información que se da de manera aleatoria. El parámetro más importante en esta operación es la tasa de recombinación (CR), cuyo valor define el porcentaje de la población generada con este operador, y en contraste con los Algoritmos Genéticos debe ser pequeño. Para este trabajo se usó el método de cruza binomial y se le asignó a CR un valor de 0.2; se eligieron 2 padres al azar y se generó 1 hijo mediante la ecuación 2.

$$Hijo_i = \begin{cases} Padre1_i \; Si \; rand \in (0,1) \geq 0.5 \\ Padre2_i \qquad En \; caso \; contrario \end{cases} \tag{2}$$

2.4 Selección

Cada vez que un individuo x_i' fue generado, se evaluó su modelo de velocidades para determinar si formaría parte de la siguiente generación o no. Debido a que el problema en cuestión es de minimización, si el valor de la evaluación $f(x_i')$ era menor que el de su padre $f(x_i)$ este individuo pasaba a la siguiente generación; de lo contrario, el padre era elegido $f(x_i)$.

2.5 Función de evaluación

La función de evaluación de este algoritmo consistió en medir los tiempos residuales (dt, que es la diferencia entre el tiempo de llegada del rayo sísmico a cada geófono calculado y el tiempo observado). Para obtener el tiempo de llegada de cada uno de los geófonos se realizó una simulación de la propagación del frente de onda dentro del volumen de estudio, empleando el modelo de [12]. Se leyeron los datos de las ubica-

ciones de las 7 fuentes de energía, de los cientos de receptores distribuidos y de los tiempos de llegada de la onda sísmica, en base a un experimento de campo realizado por un geólogo. Para calcular el tiempo de llegada dentro de la celda donde se encontraba el geófono, se usó una interpolación trilineal, utilizando los tiempos de llegada del frente de onda a los ocho vértices de la celda. Estos dos procesos son parte del ATS propuesto por [12,13]. De acuerdo a los valores iniciales de la velocidad y la profundidad (representadas en cada uno de los individuos), se generó mediante el método de Vidale [12] un modelo inicial de tiempos de llegada del frente de onda y a continuación se trazaron las trayectorias de los rayos sísmicos, siguiendo la dirección del gradiente de frente de onda en cada celda, desde el receptor a la fuente. Después de que se generaron los tiempos de viaje, se calculó el tiempo residual mediante la diferencia del tiempo observado y el calculado. Una vez realizadas estas operaciones para cada rayo que atravesaba el modelo desde el geófono hasta la fuente, se repitió el proceso para el resto de las fuentes y de los receptores y se obtuvo el valor cuadrático medio (RMS, del inglés, Root Mean Square), ver la Fig. 4.

```
Leer GeofonosxFuente, tiempos [414414]
NoFuente=1;
while NoFuente ≤ TotalFuentes do
    Leer posicionNoFuente
    NoGeofono=1;
      while NoGeofono≤TotalGeofonos do
        Leer posicionNoGeofono, t∘
        Hallar la celda que contiene NoFuente
        Obtener tc  (Interpolación con tiempos [8])
        dt=posicionNoGeofono - posicionNoFuente
        nCeldas=0;
        rayo=posicionNoGeofono;
        while rayo≠posicionNoFuente do
            Calcular gradiente del rayo en la celda
            Encontrar el paso para cambiar de celda
            nCeldas=nCeldas + 1;
              if rayo=posicionNoFuente then
                Obtener longitud del rayo
                  for i=1 hasta nCeldas do
                    Obtener trayectoria del rayo
                  end for
              end if
          end while
          NoGeofono=NoGeofono + 1;
      end while
      NoFuente=NoFuente + 1;
end while
Regresa dt
```

Fig. 4. Algoritmo de cobertura de rayos

El pseudocódigo que se presenta en la Fig. 4 muestra el funcionamiento del algoritmo de cobertura de rayos sísmicos, el cual se ejecuta después de haber calculado los tiempos de llegada, mediante la simulación del frente de onda.

3. Paralelización del algoritmo de cobertura de rayos e integración de la ED en la GPU

El algoritmo de cobertura de rayos calcula el tiempo de llegada del frente de onda desde las fuentes de energía hacia cada uno de los geófonos localizados en una capa cercana a la superficie terrestre a evaluar (tc), y halla la diferencia entre éste y el tiempo observado en el experimento (to), además verifica que, según el modelo de velocidades propuesto por la ED, cada uno de los rayos sísmicos llegue hasta los geófonos, cruzando un conjunto de celdas internas del modelo. En el algoritmo secuencial, el tiempo de llegada al geófono, así como el cálculo de la trayectoria entre el geófono y el shotpoint es realizado geófono por geófono, ralentizando el proceso de cálculo, debido a que éste debe terminar antes de empezar con el siguiente.

En este trabajo se hizo la distribución del cálculo de dt, y la trayectoria de los rayos sísmicos, de manera que cada geófono se procesara sobre un hilo diferente en una GPU NVIDIA GeForce GT 430, de 96 núcleos. Se creó la mínima cantidad de hilos posible dentro de cada bloque con el propósito de que se llevara a cabo una ejecución efectivamente paralela, debido a que en la arquitectura SIMT (Single Instruction, Multiple Thread) se ejecutan los hilos en grupos paralelos de 32 llamados warps [15]. El modelo paralelo usado en este trabajo de investigación se muestra en la Fig.5, donde cada hilo realiza el proceso que se muestra en el algoritmo de la Fig. 4.

La información correspondiente a las ubicaciones de las fuentes de energía y de los receptores, así como la de los tiempos de llegada fue transferida desde la CPU a la GPU, con la finalidad de tener acceso a ésta de manera más rápida.

En la CUDA [16] los hilos se encuentran dentro de bloques. El número de bloques para este trabajo fue calculado mediante la ecuación 3:

$$\text{nBloques} = \left[\frac{\text{N}}{\text{NMAXHILOS}}\right] \tag{3}$$

Donde N representa a los 4440 geófonos que fueron evaluados en total, ya que si bien es cierto que el número de receptores distribuidos en el volumen fue de 793, no todos pueden ser considerados en los 7 shotpoints y tampoco tuvieron la misma información con respecto a las fuentes de energía, debido a sus distintas ubicaciones. NMAXHILOS identifica al número de hilos por bloque, que puede ser desde 32 hasta 512 o 1024, dependiendo del modelo de la GPU. En este trabajo se usó NMAXHILOS= 64, para un mejor rendimiento.

Como puede notarse en la ecuación, el número bloques es el entero inmediato superior al resultado de la división debido a que, como pueden variar el número de receptores y también el número de hilos por bloque, es posible que el número de bloques resultante de esta operación no sea suficiente para ejecutar los hilos que sean necesarios, por lo que realizando el cálculo de esta manera no se dará este caso; por el

contrario, es posible que sobren, pero cuando suceda a estos se les asigna un valor de 0 para ser considerados en el proceso pero sin afectar el resultado final.

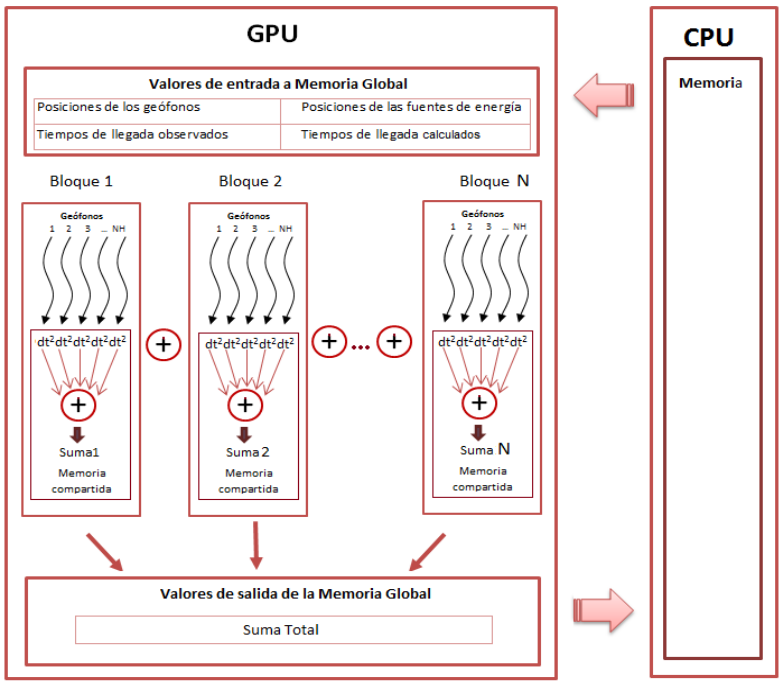

Fig. 5. Modelo de paralelización para algoritmo de cobertura de rayos

La función que ejecuta un hilo (kernel) incluye: el valor total de geófonos; total de fuentes de energía número; número de shotpoint que puede ser omitido en el experimento, pero es tomado en cuenta asignando 0 a los tiempos de los hilos que evalúen los geófonos de este número de fuente, para no afectar el proceso paralelo y el resultado final; las ubicaciones de las 7 fuentes de energía, en amplitud, longitud y profundidad del volumen de estudio; los identificadores de los geófonos; los tiempos de llegada de las ondas sísmicas, desde las 7 fuentes de energía hacia cada uno de los geófonos observados por el geólogo; los tiempos de llegada desde cada una de las fuentes de energía hacia todos los puntos del modelo 3D, calculados en la simulación del frente de onda; el resultado de la suma de los tiempos residuales al cuadrado (dt^2) de todos los geófonos; las posiciones del primero y último geófono que se evaluaron en cada fuente (ver tabla 1), debido a que los 4440 receptores fueron almacenados dentro de un vector unidimensional, además de que cada fuente genera un frente de onda diferente; es decir, 7 * 414 414 tiempos de llegada como se muestra en la Fig. 6, y son transferidos a la GPU. Cuando el algoritmo calcula el gradiente dentro de una celda, necesita los 8 tiempos de llegada de los vértices de esa celda; es por ello que el

kernel debe saber a qué fuente pertenece para realizar el cálculo de la posición en dicho tiempo.

Tabla 1. Posiciones de inicio y fin de geófonos, para cada fuente de energía

Inicio	0	488	1105	1839	2436	3124	3792
Fin	487	1104	1838	2435	3123	3791	4439

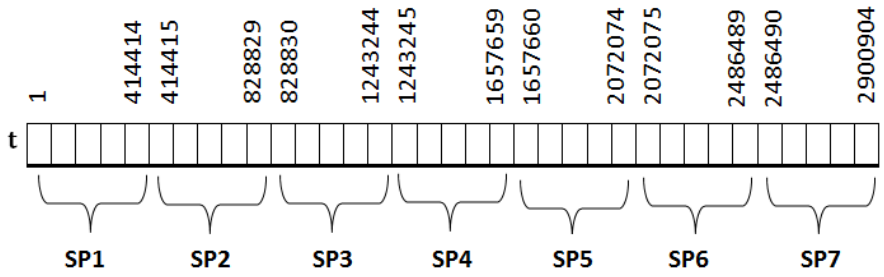

Fig. 6. Acceso al vector de tiempos

En este trabajo de investigación la memoria compartida fue usada para realizar la suma de los tiempos residuales de los geófonos en un bloque; por lo que se usó un arreglo de tipo *shared* dentro del kernel, cuyo tamaño estuvo en función del número de hilos por bloque que decidieron usarse (NMAXHILOS); en este caso se tomó el 64 debido a que tuvo un mejor desempeño en comparación con 16, 32, 128, 256 y 512.

Es importante mencionar que fue necesario un identificador del geófono dentro de cada bloque, y este se calculó con la ecuación 4:

$$I = NMAXHILOS \times blockIdx.x + threadIdx.x \quad (4)$$

Donde blockIdx.x es el identificador del bloque y threadIdx.x es el identificador del hilo dentro de cada bloque.

Pudo haber identificadores de geófonos que sobrepasaran el número total de receptores a evaluar, en consecuencia éstos no debían procesar ningún dato. Si este era el caso, al igual que se hizo con la fuente de energía omitida, automáticamente se le asignó el valor de 0 al resultado de dt, que almacenó el tiempo residual cuadrático.

Para la evaluación de cada hilo y debido a que la información de cada geófono era distinta, de acuerdo al número de fuente; se debió calcular, en base a I y a SP, el valor que se usara para desplazamiento en el vector que contiene los 414 414 tiempos de llegada de todas las fuentes de energía; con la finalidad de acceder al valor correspondiente para el hilo en ejecución. Este proceso se realizó verificando si I estaba dentro de los rangos contenidos entre el primer valor y el último de cada SP. Una vez que

todos los hilos ejecutaron sus procesos para encontrar el tiempo residual, se aseguró que todos terminaran para continuar con los demás cálculos.
Posteriormente, se realizó la suma de los valores de dt^2 de todos los hilos en cada bloque, y se almacenaron en sum. El proceso lo llevó a cabo sólo uno de los hilos y en este trabajo fue el threadIdx.x==0. La ecuación 5 se usó para llevar a cabo este proceso.

$$sum_b = \sum_{i=1}^{NMAXHILOS-1} dt^2 \quad (5)$$

Debido a que se hizo uso de datos almacenados en memoria compartida, el acceso a ellos fue mucho más rápido en comparación con la memoria global.
Para finalizar la función kernel, se hizo la suma total de los tiempos residuales pero de todos los bloques, haciendo uso de la ecuación 6:

$$\text{temp2} = \sum_{b=0}^{nBloques} sum_b \quad (6)$$

Donde nBloques es el número total de bloques, sum_b contiene la suma de los hilos en cada bloque, y $temp^2$ la suma de todos los bloques.

El kernel fue ejecutado de manera paralela en cada hilo de CUDA, devolviendo la suma total de los valores de dt^2.

4. Resultados

El algoritmo paralelo fue ejecutado 5 veces con poblaciones de 10, 20, 30, 40 y 50 individuos en cada una, sobre una tarjeta NVIDIA GeForce GT 430, la cual tiene 96 núcleos CUDA.

Las versiones secuencial y paralela fueron evaluadas de acuerdo al tiempo que tardaron en realizar el cálculo de la suma total de la diferencia al cuadrado de los tiempos de llegada a cada uno de los geófonos, con la restricción de que se generara un rayo sísmico entre la fuente y cada receptor.

La medición de tiempos en una versión secuencial se hace generalmente en la CPU; sin embargo, para efectos de igualdad en cuanto a las unidades de una medida únicamente, el algoritmo secuencial se ejecutó en un hilo dentro de un bloque sobre la GPU; no obstante, debido a que en CUDA la GPU tiene un tiempo máximo de cómputo por proceso, la tarjeta no soportó la ejecución de la versión secuencial para los 4 440 geófonos puesto que el tiempo requerido era mayor al permisible, por lo cual sólo fue posible registrar el tiempo de 1 070 de ellos antes de que la computadora terminara el proceso. Mediante el método regresión lineal se buscó un modelo en 2D que aproximara los valores de los receptores registrados para posteriormente calcular la tendencia de estos datos para 4 440 geófonos.

En la Fig. 7 puede apreciarse que, el comportamiento del tiempo tiende a incrementarse a medida que el número de geófonos aumenta. Según el cálculo realizado en este modelo, para 4 440 geófonos el tiempo estimado es de 8.9196 segundos.

En la Fig. 8 se graficaron los tiempos de ejecución del algoritmo de cobertura de rayos, en horas, en función del tamaño de población por cada versión. Puede obser-

varse que, el tiempo en la versión secuencial incrementa de manera constante, en contraste con la versión paralela donde se aprecia que el valor de la pendiente disminuye mientras el tamaño de la población crece, por lo que se deduce que si el tamaño de la población sigue incrementándose las líneas de la gráfica se separarán mucho más, remarcando la diferencia entre ambas versiones, donde la versión paralela superará en gran medida a la versión secuencial.

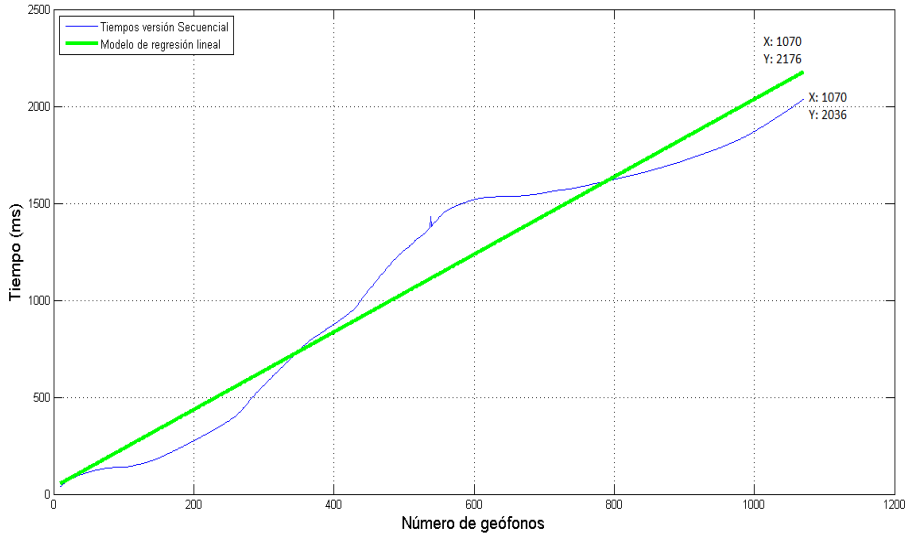

Fig. 7. Modelo de Regresión Lineal para la medición de tiempos en la versión secuencial

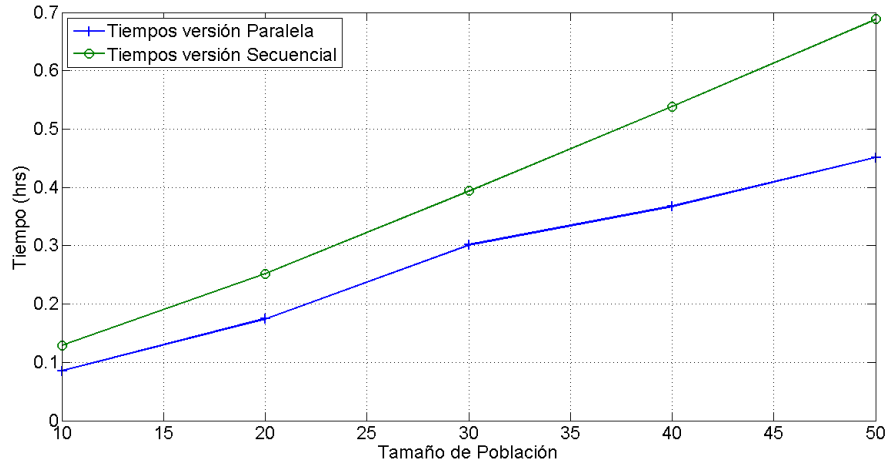

Fig. 8. Comparación de tiempos de la Versión Secuencial y Paralela

En la Fig. 9 se aprecia la gráfica con los mínimos valores de aptitud por tamaño de población. Como puede observarse, las aptitudes que corresponden al tamaño pobla-

ción 40 son, de cierta manera, las mejores en comparación con el resto de las poblaciones.

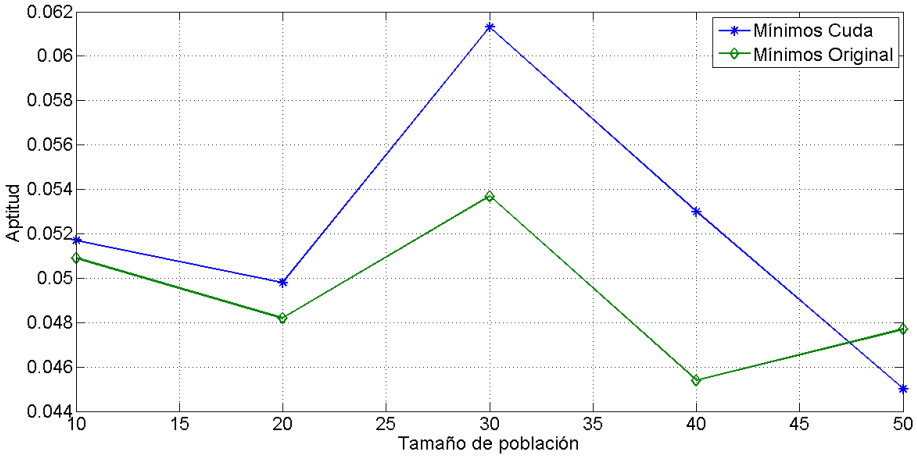

Fig. 9. Aptitud mínima por tamaño de población

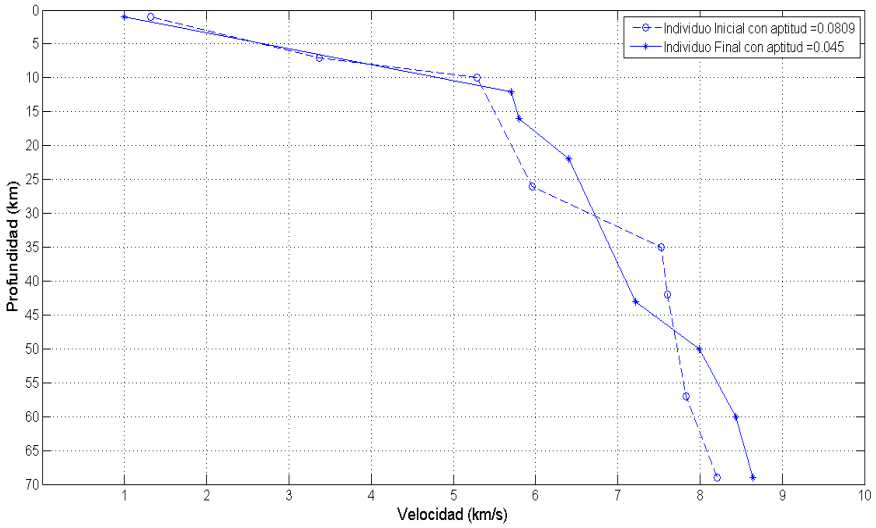

Fig. 10. Modelo de velocidades inicial con tamaño de población 40

En general puede notarse que, de acuerdo al tamaño de la población, la aptitud mejora en ambas versiones, a través de 10 generaciones y tiende a ser similar al final de cada evaluación; esto debido a que los procesos llevados a cabo son iguales en su forma secuencial y paralela.

La Fig. 10 muestra el modelo generado por la ED e indica la velocidad a la que se propagan las ondas sísmicas en cada una de las capas de los 69 km de profundidad en

el volumen de estudio. Este modelo fue obtenido por el individuo inicial y el individuo final de la versión en CUDA, con un tamaño de población de 40 individuos.

5. Conclusiones y trabajos futuros

En este trabajo de investigación se presentó la implementación de una Evolución Diferencial, donde la función objetivo es un algoritmo de cobertura de rayos sísmicos, cuya ejecución se llevó a cabo en una GPU. Si bien es cierto que la paralelización de los algoritmos evolutivos puede aplicarse en todo el proceso (generación de población inicial, operadores de selección, mutación y cruza, y la función de evaluación) la mayoría de los trabajos, incluyendo este, se enfocan únicamente a la paralelización de la función de evaluación debido a que es la que consume mayor tiempo de cómputo.

El modelo paralelo diseñado en este trabajo genera 7 frentes de onda (uno por cada fuente de energía), de tal manera que los 4 440 geófonos acceden simultáneamente a los tiempos de llegada. Esta solución mostró un mejor desempeño en comparación con la versión secuencial original al distribuir la evaluación de los geófonos de manera simultánea, en lugar en hacerlo uno detrás de otro.

La parte de mayor relevancia en este trabajo es el diseño del modelo paralelo, donde se asignó la evaluación de un geófono en cada hilo de CUDA y se formaron grupos de la mínima cantidad posible de hilos por cada bloque ya que la ejecución de manera simultánea de éstos dentro de un bloque se da en conjuntos de 32 hilos; y aunque este valor representa la agrupación de hilos más pequeña, en este caso se usaron 64 ya que este número mostró un mejor desempeño para la reducción del tiempo de ejecución del algoritmo de cobertura de rayos, al ser procesado sobre la GPU.

El desarrollo de este trabajo muestra la solución a uno de los módulos que conforman un ATS; sin embargo existen otros procesos tales como la generación del frente de onda y el suavizado de la propagación de rayos sísmicos, que serán paralelizados más adelante.

Otra propuesta es usar otros algoritmos de búsqueda u optimización como las estrategias evolutivas, o los algoritmos bioinspirados.

Por la parte de tomografía sísmica, se planea diseñar otro modelo paralelo para aprovechar la memoria compartida, el cual no realice el trazo de los rayos siguiendo la trayectoria de la onda celda por celda, sino que cada una de las celdas haga un monitoreo de los rayos que pasan por ellas, de tal manera que un hilo represente una celda y los 8 tiempos de llegada, del shotpoint a sus vértices, estén almacenados en la memoria compartida.

Referencias

1. Lee, E.-J., Huang, H., Dennis, J. M., Chen, P., and Wang, L.: An optimized parallel algorithm for seismic tomography. Computers Geosciences (2013)
2. Ramírez Cruz, J., Fuentes, O., Romero, R., and Velasco, A.: A Hybrid Algorithm for Crustal Velocity Modeling. In: Batyrshin, I. and Mendoza, M., edi-

tors, Advances in Computational Intelligence, volume 7630 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 329–337 (2013)
3. Oiso, Masashi and Matsumura, Yoshiyuki and Yasuda, Oshiyuki and Ohkura, Kazuhiro,:Implementing genetic algorithms to CUDA environment using data parallelization. Technical Gazette, Hrcak Portal of scientific journals of Croatia,, vol.18, pp. 511– 517 (2011)
4. Kirk, D. B. and Hwu, W.-m. W. : Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann Publishers Inc., 1st edition, (2010).
5. Mu, D., Chen, P., and Wang, L.: Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using multiple GPUS with CUDA and MPI. Earthquake Science, vol. 26, no. 6, pp. 377–393 (2013)
6. Chitty, D. M., Malvern, Q., and Ps, W.: A data parallel approach to genetic programming using programmable graphics hardware. In: GECCO 07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press, pp. 1566–1573 (2007)
7. González, S. J. D.: Implementación de un algoritmo de evolución diferencial paralelo basado en unidades de procesamiento gráfico. Master's thesis, Universidad Michoacana de San Nicolás de Hidalgo (2011)
8. Sun, C., Zhou, H., and Chen, L.: Improved differential evolution algorithms. In: Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on, vol. 3, pp. 142–145 (2012)
9. Gao-yang, L. and Ming-guang, L.: The summary of differential evolution algorithm and its improvements. In: Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, vol. 3 (2010)
10. Bujok, P. and Tvrdik, J.: Parallel migration models applied to competitive differential evolution. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 13th International Symposium on (2011)
11. Ao, Y. and Chi, H.: Experimental study on differential evolution strategies. In: GCIS '09, WRI Global Congress on Intelligent Systems (2009)
12. Vidale, J. E.: Finite-difference calculation of travel times in three dimensions. Geophysics, vol. 55, no. 5, pp. 521–526 (1990)
13. J. A. Hole: Nonlinear high-resolution three-dimensional seismic travel time tomography. Journal of Geophysical Research, vol. 97, no. 85, pp. 6553–6562 (1992)
14. Engelbrecht, A.: Computational Intelligence: An Introduction. Wiley, pp. 242–244 (2007)
15. NVIDIA Corporation: NVIDIA CUDA C Programming Guide (2014)
16. Sanders, J. and Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional (2010)