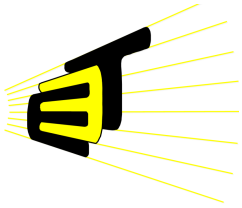
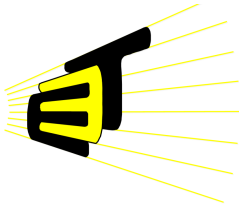


REVISIÓN Y ANÁLISIS DEL ALGORITMO DE MIGRACIÓN REVERSA EN TIEMPO (RTM) PARA SU POSTERIOR IMPLEMENTACIÓN EN UNA GPU

Jhonatan Andrés Amado Valderrama

Universidad Industrial de Santander
Facultad de Ingeniería Físico-Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2014




REVISIÓN Y ANÁLISIS DEL ALGORITMO DE MIGRACIÓN REVERSA EN TIEMPO (RTM) PARA SU POSTERIOR IMPLEMENTACIÓN EN UNA GPU

Jhonatan Andrés Amado Valderrama

Trabajo de investigación presentado como requerimiento para optar al titulo de:

Ingeniero Electrónico

Tesis desarrollada dentro del programa 0266-2013

Director
Ph.D(c) William Alexander Salamanca

Codirector:
Ph.D Flor Alba Vivas

Universidad Industrial de Santander
Facultad de Ingeniería Físico-Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2014

Tabla de Contenido

Lista de Tablas

Lista de Figuras

RESUMEN

Titulo:

Revisión y Análisis del Algoritmo de Migración Reversa en Tiempo (RTM) para su Implementación en una GPU [1]

Autor: Jhonatan Andrés Amado Valderrama [2].

Palabras Claves: RTM, CPU, GPU, CUDA

El siguiente documento presenta la implementación del algoritmo de migración reversa en tiempo RTM en una GPU nvidia usando el lenguaje de programación C-CUDA.Compute Unified Device Architecture es el lenguaje de programacion que permite realizar un modelo de programacion heterogenea (uso de CPU y GPU para solucionar el problema de investigacion). El uso de las GPUs para implementar este tipo de algoritmos es debido a su alto costo computacional si estos son ejecutados con el tipo programación secuencial.

La sísmica de reflexión es una técnica muy utilizada en la exploración geofísica ya que permite obtener información del subsuelo adquiriendo los tiempos de arribo de las ondas generadas artificialmente mediante cargas explosivas o algún otro método mecánico para generar dichos perturbaciones. El objetivo es obtener información del terreno, referente a la estructura interna, propiedades dinámicas del suelo, disposición geométrica de las diferentes capas, para así extraer una imagen del subsuelo.

Cuando se realiza el procesamiento de datos para generar las imágenes de las capas geológicas con los diferentes tipos de migración, los resultados obtenidos varían de acuerdo al tipo de método migratorio usado. RTM es un tipo de migracion en profundidad que da una mejor calidad de imagen, una mejor relación señal al ruido, y una delineación más clara de las estructuras geológicas de alta complejidad.

[1] Proyecto de Grado

[2] Facultad de Ingenieria Fisico Mecanicas. Escuela de Ingenieria Electrica, Electronica y Telecomunicaciones. Director: Ph.D(c) William A. Salamanca Becerra. Codirector Ph.D Flor Alba Vivas.

ABSTRACT

Title:

Review and analysis of Reverse-Time Migration (RTM) algorithm for a GPU implementation [1]

Author: Jhonatan Andrés Amado Valderrama [2].

Keywords: RTM, CPU, GPU, CUDA

The following paper presents the implementation of the reverse time migration (RTM) algorithm for nvidia GPU using the programming language C-CUDA. Compute Unified Device Architecture is the programming language that allows heterogeneous programming model (use of CPU and GPU to solve the problem of research). Using GPUs to implement such algorithms is due to its high computational cost if they are executed with sequential programming type.

The seismic reflection is widely used in geophysical exploration because it allows acquiring subsurface information arrival times of waves generated artificially by explosives or some other mechanic method technique to generate this disturbances. The objective is to obtain information concerning the internal structure, dynamic soil properties, geometric arrangement of the different layers, to take an image of the subsurface.

When data processing is performed to generate images of geological layers with different types of migration, the results vary according to the type of migration method used. RTM is a depth type migration which gives a better picture quality, better signal to noise ratio, and a clearer delineation of high complexity geological structures.

[1] Thesis

[2] Faculty of Physics Mechanics Engineering. Electrical, Electronics Engineering and Telecommunications School. Advisor: Ph.D(c) William A. Salamanca Becerra. Co-Advisor Ph.D Flor Alba Vivas.

Revisión y Análisis del Algoritmo de Migración Reversa en Tiempo (RTM) para su posterior Implementación en una GPU

Jhonatan Andrés Amado Valderrama

Universidad Industrial de Santander

Resumen

La implementación en CPU y GPU del código de Migración Reversa en Tiempo 2D es presentada a continuación. Se realiza un especial análisis en la implementación de la solución discreta de la ecuación de onda acústica 2D con CUDA sobre una GPU de arquitectura Fermi, y la implementación de PML sobre las fronteras. La aplicación, dependencias y visualización es realizada con el programa Seismic Unix. Diferentes formas en el manejo de los datos son planteados y se realizan comparaciones para el análisis del rendimiento en diferentes arquitecturas.

Palabras Claves: RTM, CPU, GPU, CUDA

INTRODUCCIÓN

La búsqueda de reservas petroleras en áreas de alta complejidad geológica, como se observa en la figura 1 , se lleva a cabo por medio de un proceso que va desde el diseño de la adquisición[1], pasando por una etapa de preprocesamiento de las trazas (deconvolución, filtros), siguiendo por un proceso iterativo que va desde el análisis de velocidades RMS, pasando por una migración sísmica en tiempo para obtener un modelo de velocidades en profundidad y finalmente aplicar la Migración en Profundidad RTM[2] para la posterior interpretación del mapa de reflectividad por parte de un ojo experto. RTM, es un proceso de alto costo computacional debido a que debe solucionar, para cada punto del modelo de velocidades, dos campos de onda (campo de la fuente y el campo de los receptores) y aplicar la condición de imagen. Debido a que es un proceso que puede demorar meses, computacionalmente en el proceso de ejecución es inviable usando los métodos de programación estándar que se conocen (implementación secuencial de los códigos). Por ello se debe implementar usando programación en paralelo.

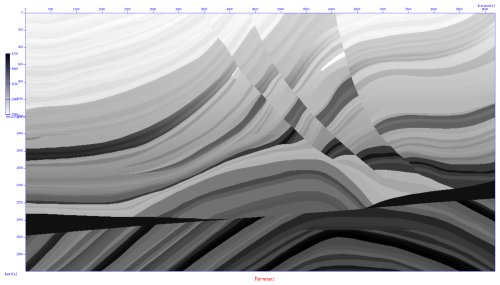


Figura 1: Modelo del subsuelo de alta complejidad geológica. Adaptado de [5].

En la última década, las GPUs [3] han evolucionado, no solo para los consumidores en el diseño gráfico, sino también para acelerar apli-

[1]Una adquisición con datos reales puede tener un dt=4 [ms] y un tiempo de adquisición de T= 9 [s] para un total de muestras a procesar de nt=2250

[2]RTM: Migración Reversa en Tiempo

[3]GPU: Unidad de Procesamiento Gráfico

caciones científicas a un costo razonable, usando una interfaz software-hardware llamada CUDA[4]. Este desarrollo permite acortar el tiempo de ejecución, aplicando el paradigma de la programación en paralelo.

MIGRACIÓN REVERSA EN TIEMPO 2D

Los métodos clásicos de migración para áreas geológicas complejas se ven desafiados ya que el resultado del mapa de reflectividad no tiene suficiente resolución, o el método no es capaz de resolver algunos eventos sísmicos, produciendo errores en la localización de yacimientos de hidrocarburos, ya que estos suelen estar en áreas donde los buzamientos[5] alcanzan hasta los 90 grados como se aprecia en la figura 2. La representación de una imagen correcta de yacimientos bajo los cuerpos salinos es un nuevo reto en el procesamiento, ya que cuando el frente de onda alcanza estas áreas, las ondas reflejadas son casi horizontales, por lo que se necesitan técnicas especializadas para reconocer y migrar ese tipo de ondas; además las estructuras salinas, aparte de tener cambios de velocidades abruptos, en la cima de ellas su estructura es muy irregular lo cual dispersa las ondas sísmicas en distintas direcciones. A menos que se realice un proceso que sea capaz de acumular y reconstruir toda esa energía dispersa, la información en la parte superior del domo salino se perderá, obteniendo un mapa de reflectividad incorrecto.

El algoritmo de RTM es la solución para estas estructuras geológicas ya que permite que las ondas se propaguen en todas las direcciones. La ventaja de RTM frente a los otros métodos de migración es su especial habilidad de producir imágenes en áreas de cambios abruptos de velocidades horizontales, sin importar los buzamientos que tenga dicha estructura. Esto es posible ya que este algoritmo resuelve la ecuación de onda hacia adelante en el tiempo para la fuente, y hacia atrás en el tiempo para los receptores, para luego aplicar una correlación entre estos dos campos.

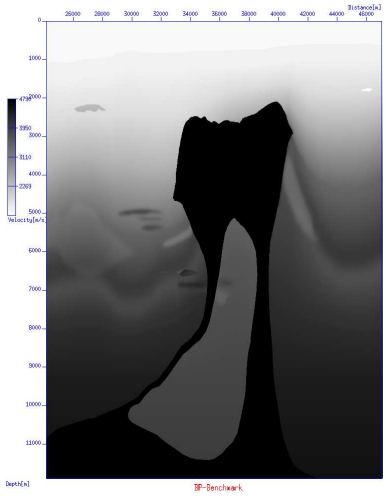


Figura 2: Modelo del subsuelo de alta complejidad geológica con buzamiento.Estas variaciones en las velocidades son las representativas en el Golfo de México. Adaptado de [1].

1. PROGRAMACIÓN EN PARALELO USANDO CUDA

El uso de las GPUs para acelerar las operaciones de cálculo científico o cálculos de propósito general se le conoce como GPU-*computing*. Si en una plataforma heterogénea, se tiene CPU+GPU, esta combinación optimizaría el procesamiento de datos, ya que se tiene una unidad de procesamiento optimizada para la latencia, y otra optimizada para el *throughput*. De esta forma, al desarrollar un código con CUDA, partes de éste se ejecutarán en la CPU mientras que las que demanden cómputo intensivo serán ejecutadas en GPU.

La plataforma de software-hardware CUDA creada por NVIDIA ofrece extensiones en lenguaje de programación C , C++ Python,

[4]CUDA: Dispositivos de Arquitectura de Computo Unificada

[5]Buzamiento: Ángulo que forma la línea de máxima pendiente de una superficie (estrato, capa, filón o falla) con su proyección sobre el plano horizontal.

entre otros, que permiten implementar operaciones críticas en una GPU. Utilizando CUDA, las partes críticas del código, se podrán ejecutar en cientos de núcleos que tiene una GPU. Este tipo de implementaciones, cuando la segmentación de un código es la adecuada, y la elección de las ejecuciones críticas están bien desarrolladas para aprovechar la máxima cantidad de recursos de una GPU, disminuirá el tiempo de ejecución con respecto al que sería empleado al usar solo CPU.

Al elegir la aplicación a ejecutarse en paralelo, el primer reto que se tiene, es entender la jerarquía en la ejecución de la aplicación, ya que se tienen dos perspectivas. La perspectiva del hardware, y la perspectiva del programador.

1.1. PERSPECTIVA DEL HARDWARE.

La arquitectura de una GPU está basada en varios arreglos de SM[6] como se aprecia en la figura 3. En cada SM se encuentra un número determinado de núcleos CUDA, su respectiva jerarquía de memoria, registros, unidades de carga y almacenamiento, unidades de funciones especiales y el administrador de warps. Cada SM de una GPU está diseñado para que ejecute la cantidad de hilos asignados en el mismo instante de tiempo. Y ya que se pueden tener varios SMs por GPU, la cantidad de hilos que se ejecutarán concurrentemente será superior. Cuando un *kernel*[7] es lanzado, éste, se "mapea", a lo largo de la cantidad de SMs que existan en la GPU.

1.2. PERSPECTIVA DEL PROGRAMADOR

En la perspectiva del programador, las unidades básicas, serán los *hilos*, *bloques* y *arreglos de bloques*. Cada una de estas unidades podrá ser organizada en un arreglo unidimensional bidimensional o tridimensional, y estos tipos de arreglos, estarán limitados por la arquitectura en la que se ejecutará la aplicación. Cuando un *kernel* es lanzado en la GPU, éste estará definido por dos parámetros. La cantidad de *ThreadBlocks*[8] y la cantidad de *Grids*[9]. Esta información debe ser congruente con las limitaciones de hardware que existan.

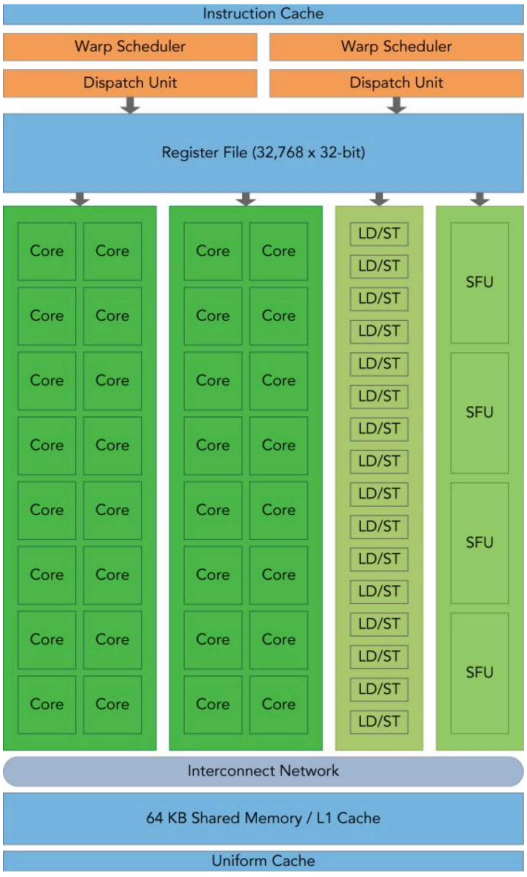


Figura 3: Streaming Multiprocessors para la arquitectura Fermi. Tomado de [8]

Cuando se desarrolla una aplicación que se ejecutará en una GPU, es necesario entender ambas perspectivas, ya que no se puede programar ignorando las limitaciones existentes por el hardware. Para ello, en cuanto a la jerarquía en la ejecución de la aplicación se tiene

[6] SM: *Streaming Multiprocessors.*
[7] kernel: *Tipo de funcion que se ejecutara en la GPU*
[8] *ThreadBlocks: hilos por bloque*
[9] *Grids: Arreglos de bloques*

la siguiente regla:

Un *threadBlock* es asignado a un solo SM. Una vez éste es asignado, permanecerá ahí, hasta que la ejecución haya terminado. A un SM le pueden ser asignados mas de un *threadblock*. [4]

Esta relación se ve en la figura 4.

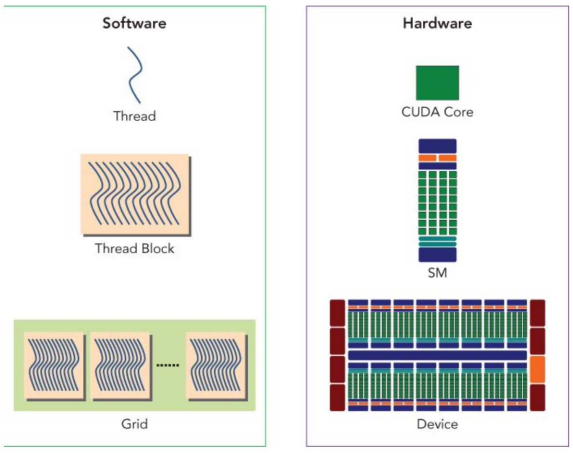


Figura 4: Equivalencia perspectiva programador y hardware. Tomado de [4].

Vinculando ambas perspectivas para la GPU GTX 580 donde se implementará RTM2D, se presenta la Tabla 1.

Elem	Perspectiva Lógica	Perspectiva Física
Thread	Max=1024 Arreglos(x,y,z) (1024,1024,64)	Num SP=32 Cada SP ejecuta un Thread al mismo tiempo
Block	Arreglos(x,y,z) (65535,65535,65535)	Num SM=16
Kernel	–global–	Solo un Kernel al tiempo es lanzado en la GPU

Tabla 1: Valores en la GPU GTX-580

2. IMPLEMENTACIÓN DE LA ECUACION DE ONDA ACUSTICA EN CUDA

Los fenómenos físicos ondulatorios que acontecen durante la propagación de la energía al momento de realizar las adquisiciones sísmicas pueden ser descritos por la siguiente ecuación diferencial:

$$\frac{\partial^2 P}{\partial t^2} = C^2 \Delta P \tag{1}$$

La ecuación número 1 es conocida como la Ecuación de Onda Acústica temporal. Esta ecuación soluciona la dinámica con densidad constante de las ondas en función de un campo de presión $P(x, z, t)$, donde x, z son coordenadas espaciales y t es la coordenada temporal, mientras la velocidad del medio C por donde las ondas se transmiten cambia espacialmente, dando como resultado fenómenos ondulatorios. En el caso de la sísmica de reflexión, las ondas a analizar son las ondas tipo P las cuales son ondas longitudinales[10]. El operador laplaciano Δ representa el diferencial espacial aplicado en la solución de esta ecuación. Para obtener una solución numérica a esta ecuación diferencial, existen varios métodos, entre ellos el método de elementos Finitos (FEM), los métodos Espectrales, el método de elementos en la frontera (BEM) y las Diferencias Finitas en el dominio del tiempo (FDTD). El método utilizado en la solución de la ecuación de onda para la implementación del algoritmo de Migración Reversa en Tiempo es el de FDTD-2D (2 Orden Temporal, 8 orden espacial). La ecuación 1 discretizada bajo los criterios anteriores usando la expansión por series de Taylor y grilla homogénea ($\delta x = \delta z = \delta h$) es la siguiente:

$$p^{n-1}_{(x,z)} - 2p^{n}_{(x,z)} p^{n+1}_{(x,z)} = G^2 \Delta P \tag{2}$$

[10]Otros tipos de onda suelen ser estudiados para una más detallada representación de la dinámica de las ondas en el subsuelo, como lo son la ondas tipo S y las ondas superficiales que en este trabajo no se tendrán en cuenta.

$$G^2 = \frac{C^2_{(x,z)} * \delta t^2}{\delta h^2} \tag{3}$$

Un ejemplo de discretización del laplaciano de segundo orden para su posterior representacion en stencil es el siguiente:

$$\left.\begin{aligned} \Delta P = C_2 P^n_{(x-2,z)} + \\ C_1 P^n_{(x-1,z)} + C_0 P^n_{(x,z)} + \\ C_1 P^n_{(x+1,z)} + C_2 P^n_{(x+2,z)} + \\ C_2 P^n_{(x,z-2)} + C_1 P^n_{(x,z-1)} + \\ C_0 P^n_{(x,z)} + C_1 P^n_{(x,z+1)} \\ + C_2 P^n_{(x,z+2)} \end{aligned}\right\} \quad \Delta \text{ 2 Orden} \tag{4}$$

$$\left.\begin{aligned} \Delta P = C_4 P^n_{(x-4,z)} + \\ C_3 P^n_{(x-3,z)} + C_2 P^n_{(x-2,z)} + \\ C_1 P^n_{(x-1,z)} + C_0 P^n_{(x,z)} + \\ C_1 P^n_{(x+1,z)} + C_2 P^n_{(x+2,z)} \\ + C_3 P^n_{(x+3,z)} + C_4 P^n_{(x+4,z)} + \\ C_4 P^n_{(x,z-4)} + C_3 P^n_{(x,z-3)} \\ + C_2 P^n_{(x,z-2)} + C_1 P^n_{(x,z-1)} + \\ C_0 P^n_{(x,z)} + C_1 P^n_{(x,z+1)} \\ + C_2 P^n_{(x,z+2)} + C_3 P^n_{(x,z+3)} + \\ C_4 P^n_{(x,z+4)} \end{aligned}\right\} \quad \Delta \text{ 8 Orden} \tag{5}$$

Donde C_0, C_1, C_2, C_3, C_4 son los valores de los coeficientes para el esquema de 8 orden en el espacio, calculados por el algoritmo 1.

Data: M,N,$x_0, \alpha_0, \alpha_1, \alpha_2 ... \alpha_N$
Result: coef[$C_0 C_1$...]
$\delta^0_{0,0} = 0$;
c1:=1;
for *n=1 to N* **do**
  c2=1;
  for *v=0 to n-1* **do**
    c3=$\alpha_n - \alpha_v$;
    if $n \leq M$ **then**
      $\delta^n_{n-1,v} = 0$;
    end
    for *m=0 to min(n,M)* **do**
      $\delta^m_{n,v} = ((\alpha_n - x_0)\delta^m_{n-1,v} - m\delta^{m-1}_{n-1,v})/C_3$;
      next m;
    end
    next v;
    for *m=0 to min(n,M)* **do**
      $\delta^m_{n,v} = \frac{c1}{c2}(m\delta^{m-1}_{n-1,n-1} - (\alpha_{n-1} - x_0)\delta^m_{n-1,n-1})$;
      next m;
    end
    c1=c2
  end
  next n
end

Algoritmo 1: Cálculo de coeficientes del esquema de diferencias finitas. Tomado de [6]

Y G es el parámetro de *Courant* esencial para el análisis de estabilidad del esquema de diferencias finitas.

2.1. PRECISIÓN Y ESTABILIDAD

Al momento de realizar cálculos numéricos es fundamental conocer que tan preciso y estable es el resultado. Para ello a continuación mencionaremos las condiciones de estabilidad y precisión manejadas para la ecuación (2)

- **Precisión** Son los errores relacionados por el truncamiento de la serie de Taylor para los operadores diferenciales y errores relacionados por el redondeo en la representación digital de los valores en las operaciones.

- **Estabilidad** Dado por la consistencia en la solución por un método numérico mientras se propagan los errores.

Para la solución de los errores aportados por la precisión, se discretizó espacialmente con orden 8 , (ver figura 5), la ecuación 1 dando claras diferencias en la representación del frente de onda respecto a la discretización espacial con orden 2 . (ver figura 6), además de incluir el parámetro Δh calculado por la ecuación 6.

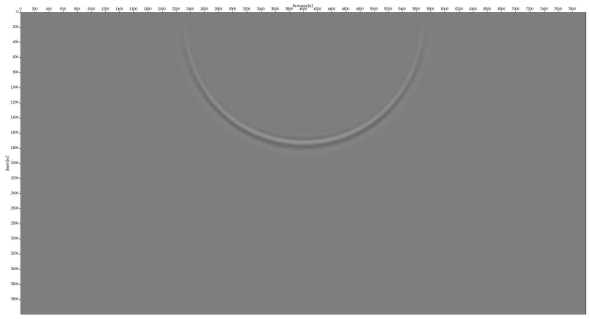


Figura 5: Ecuación de onda discretizada con orden 8 en el espacio

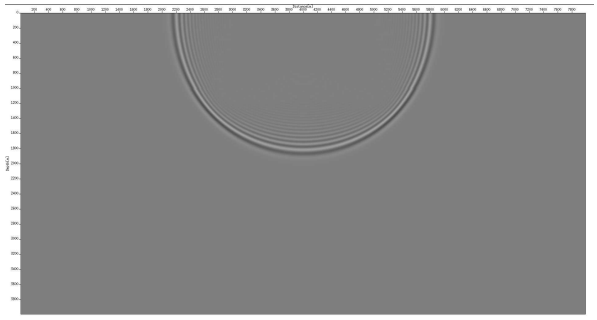


Figura 6: Ecuación de onda discretizada con orden 2 en el espacio

$$\Delta h = \frac{Cmin}{S * Fq} \tag{6}$$

donde S es el número de puntos por longitud de onda.

La condición de estabilidad para los esquemas de diferencias finitas están dados por la siguiente expresión. Adaptado de [7]

$$\frac{c\delta t}{\delta x} \leq \sqrt{\frac{\varepsilon_1}{\varepsilon_2}} \tag{7}$$

donde ε_1 es la suma de los valores absolutos de los coeficientes para el operador de las diferencias finitas temporales y ε_2 representa la suma de los valores absolutos de los coeficientes del operador laplaciano ΔP. La condición de estabilidad, despejando de la ecuación (7) es:

$$\frac{Cmax * \delta t}{\delta h} = \frac{3 * \sqrt{35}}{32} \tag{8}$$

2.2. MANIPULACION DE DATOS

El cálculo de cada punto de la grilla de velocidades esta dado por la representación en *stencil* de los datos. Ver figura 7.

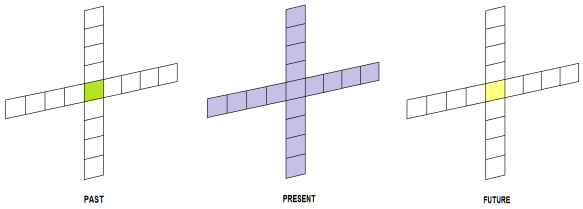


Figura 7: Dependencia de datos del *stencil*. Cada punto (x, z) de la grilla del campo futuro(amarillo) dependerá de ciertas operaciones ponderadas del campo actual(azul) dados por el orden espacial, y del punto (x, z) del campo anterior (verde).

Debido a que las GPUs no existen un sistema operativo que administre los recursos, y son dispositivos asíncronos, no hay forma de prevenir directamente los fallos en los cálculos al momentos de acceder a la memoria, por lo tanto un estricto y detallado manejo de datos debe ser utilizado.

Suponga que se debe calcular el punto verde mostrado en la figura 8. Al momento de realizar el *stencil* en el campo presente se

tiene identificación y acceso a cada punto para su cálculo. Ahora si se debe calcular el punto verde mostrado en la figura 9 el acceso a los valores de los datos representados por los puntos rojos no esta garantizado, tomando valores erróneos de esos puntos causando una falla en el cálculo del punto futuro de ese campo.

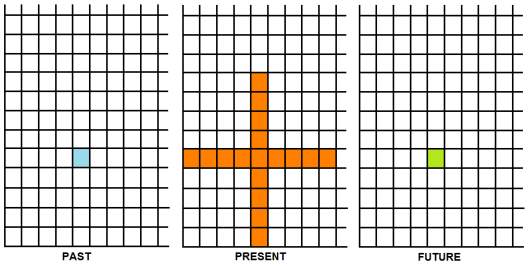


Figura 8: Cálculo del *stencil* para el punto verde sin errores en los valores

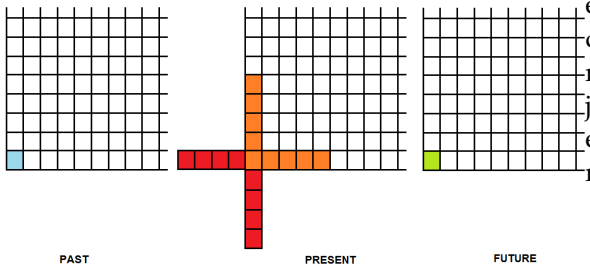


Figura 9: Cálculo del *stencil* para el punto verde con errores en acceso de los valores

Por ello se plantean las siguientes dos formas de arreglos para el manejo de los datos.

2.2.1 Manejo de datos 1: Modo "Tetris"

Este modo evita calcular los puntos que no se encuentran reservados en la memoria de la GPU. Para ello se deben establecer 9 tipos de cálculos del punto futuro del campo de onda. Ver figura 10.

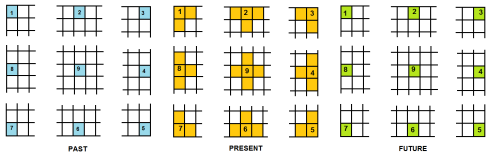


Figura 10: Cálculo de los 9 tipos de *stencil*. Este seria el ejemplo para el cálculo del *stencil* de orden 2 espacial y orden 2 temporal

2.2.2 Manejo de datos 2: Modo "0s en los límites"

En este modo, el cálculo del punto futuro del campo de onda se realiza solo con un tipo de ecuación pero no se debe tener en cuenta los datos de los límites de la grilla de velocidad. Para ello en el código se restringen los cálculos de esos puntos. Ver figura 11. La ventaja de este método en comparación con el anterior es la facilidad, legibilidad del código y sobretodo el menor número de cálculos para diferentes datos que se realizan, ya que las GPUs son mas eficientes, si para un determinado conjunto de datos (puntos de la grilla del modelo de velocidades), se le aplica sobre ellos la misma operación.

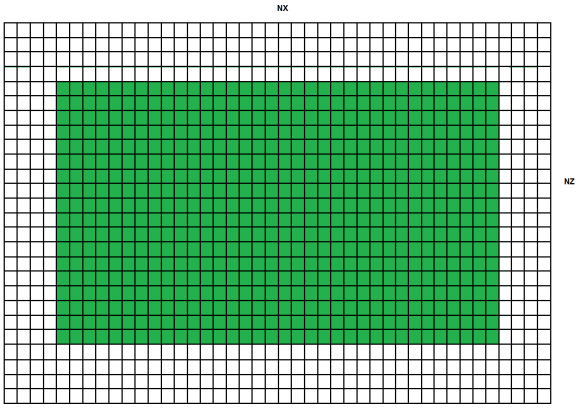


Figura 11: Manejo de datos acortando el modelo.El área sombreada representa la ubicación de los datos que pueden ser calculados.

2.3. SOLUCIÓN DE LA ECUACIÓN DE ONDA EN UNA GPU

La solución de la ecuación de onda acústica en una GPU esta dada por la implementación de 2 funciones. La primera de ellas llamada *stencil-eval*, una función que es precedida por el identificador –*global*–. Este identificador, es interpretado por el compilador *nvcc*, como una función que se ejecutará solo en la GPU. Dentro de esta función, se pasan los parámetros del modelo, el campo de velocidades, los punteros de los campos pasado, presente y futuro, y los parámetros para aplicar las condiciones de frontera. Luego se establecen los identificadores para cada *thread* en términos de cada dimensión a evaluar. Ver algoritmo 2 . Dentro del *Kernel* se limita la grilla de datos a utilizar, para que el *stencil* no acceda a datos erróneos.

Data: Modelo Velocidades, Dimensiones, punteros, otros parámetros
Result: stencil
—global— stencileval;
i=threadIdx.x + block;
k=threadIdx.y + block;
if *i and k pertenecen al campo* **then**
  id = relación i and k;
  Cálculo parametro Courant para [id];
  Ejecute la ecuación de onda discreta, 8 orden espacial, 2 orden temporal
end

Algoritmo 2: stencil-eval

La segunda función es la encargada de invocar a *stencil-eval*, reservar la memoria para la GPU, realizar todas las transferencias de datos del *host*[11] al *device* [12] , calcular el número de *threadBlock* y *Grid* necesarios dependiendo del tamaño del modelo, devolver el resultado al *host*, actualizar los campos para seguir con la siguiente iteración y liberar la memoria utilizada. Ver algoritmo 3.

[11] *Host:* Término usado en CUDA referente a la CPU
[12] *Device:* Término usado en CUDA referente a la GPU

Data: Modelo Velocidades, Dimensiones, número pasos de tiempo, otros parámetros
Result: fdtd2D
void fdtd2D;
Inicializar variables y punteros;
Reservar memoria en Host;
Reservar memoria en Device;
memset en GPU;
memcpy Modelo Velocidades;
cálculo dimGrid, dimBlock
for *todos los pasos de tiempo* **do**
  stencil-eval <<< dimGrid,dimBLock >>> (arguments);
  memcpy campo futuro al host;
  actualizar variables;
end
free (all)

Algoritmo 3: fdtd2D

2.3.1 Segmentación del problema en la GPU

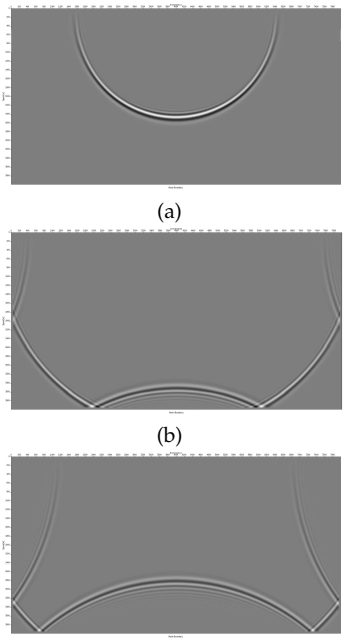
Un efecto positivo en el desempeño de la implementación y ejecución del código en una GPU, es la correcta segmentación del dominio. Para ello dos estructuras, dentro de las APIs de CUDA son utilizadas, *dimGrid* y *dimBLock*. Estas estructuras son utilizadas para pasar el número de *threadsBlock* y el número de *Grid* a ejecutarse en una GPU. Para este caso, es necesario indexar los *threads* en 2 dimensiones.

Ya que los tamaños de los modelos de velocidades varían dependiendo de la adquisición, y no es muy probable que estos sean múltiplo del tamaño de los *Blocks*, se debe buscar una expresión numérica, que permita calcular el número de bloques a utilizar. Para ello, la estructura *dimGrid, dimBLock* retorna el valor entero mayor o igual al número ingresado. La expresión numérica es la siguiente:

$$Ceil = \frac{ModelSize - 1}{BlockSize} + 1 \qquad (9)$$

2.4. FRONTERAS

Al solucionar numéricamente la ecuación de onda acústica hay un fenómeno que se debe evitar, y este es el de las reflexiones existentes en las fronteras. Ver figura 12.
Debido a que este fenómeno debe ser eliminado para un correcto modelado de la ecuación de onda, se implementaron los desarrollos propuestos por [9] y por [2], y se eligió el que proporcionara mejores resultados.



(a)

(b)

(c)

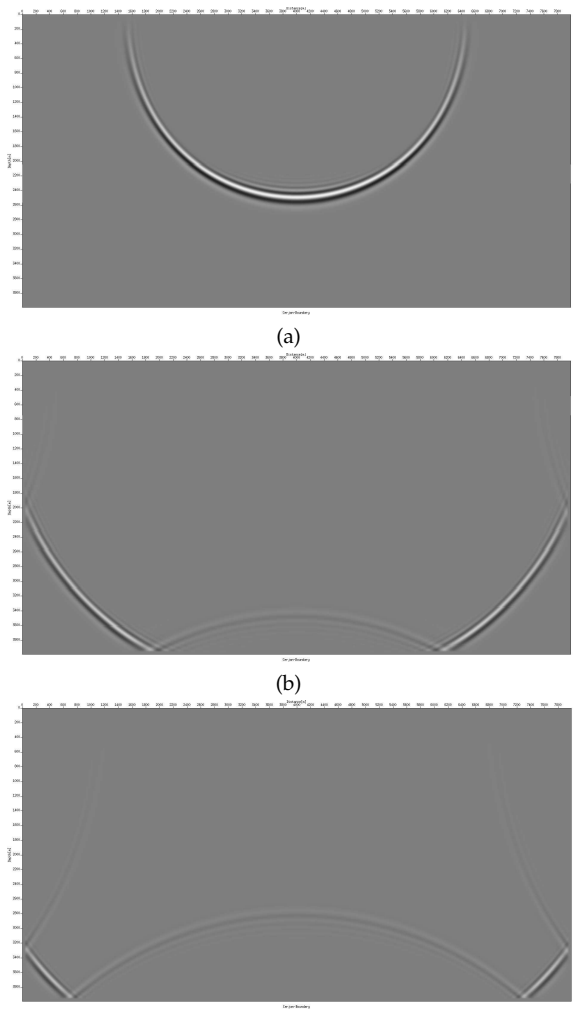
Figura 12: Dinámica del frente de onda en un medio de velocidad constante sin condiciones de frontera. Figura 12a Frente de onda 12b Frente de onda llegando a la frontera. Figura 12c Frente de onda reflejado.

2.4.1 Factor de atenuación

Tomado de [9], este método reduce las ondas reflejadas en los límites del modelo aplicando un factor de atenuación en el cálculo del valor futuro del campo dado por la ecuación 10

$$weight(i) = e^{((L-i)\times DW)^2} \qquad (10)$$

donde L representa el número de puntos en donde aplicar el factor de atenuación, y $DW(DampWeight)$ el valor numérico del factor de tenuacion . Para la implementación se tomó un $L = 20$ y $DW = 0.015$ sugeridos en [9] obteniendo los siguientes resultados. Ver (13).



(a)

(b)

(c)

Figura 13: Dinámica del frente de onda en un medio de velocidad constante aplicando condiciones de frontera Cerjan. Figura 13a Frente de onda 13b Frente de onda llegando a la frontera. Figura 13c Frente de onda reflejado.

2.4.2 CPML

Otro método para reducir las ondas reflejada debido a las fronteras del modelo es CPML. Para mitigar este fenómeno el uso de CPML esta condicionado a aplicar la combinación de los términos convolucionales en la ecuación 1. Para ello la formulación de este método esta dado por: Tomado de [10]

$$\frac{\partial^2 P}{dt^2} = v^2 \left(Px + Pz\right) \tag{11}$$

Donde Px y Pz están dados por 12 y 13

$$Px = \partial_x Ax + \psi_x \tag{12}$$

$$Pz = \partial_z Az + \psi_z \tag{13}$$

$$Ax = \partial_x p + \phi_x \tag{14}$$

$$Az = \partial_z p + \phi_z \tag{15}$$

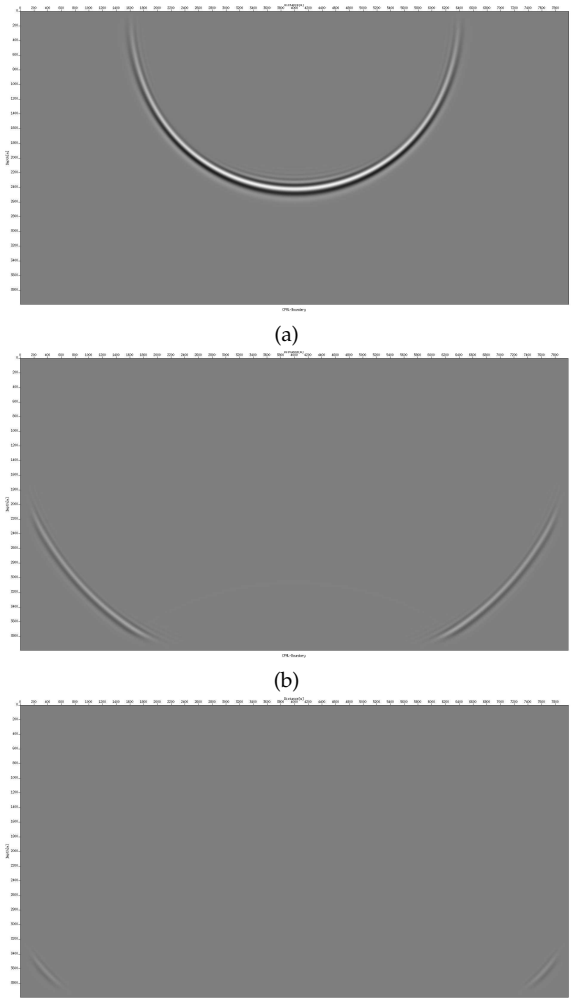
Los valores de ψ_x y ψ_z son los términos convolucionales de A_x y A_z y siguiendo la regla de la cadena, ϕ_x y ϕ_z son lo términos convolucionales de Px y Pz. Estos valores son calculados usando el siguiente arreglo:

$$\left.\begin{aligned} \psi_x^n &= b_x \psi_x^{n-1} + (b_x - 1)\, \partial_x^{n+1/2} A_x \\ \psi_z^n &= b_z \psi_z^{n-1} + (b_z - 1)\, \partial_z^{n+1/2} A_z \\ \phi_x^n &= b_x \phi_x^{n-1} + (b_x - 1)\, \partial_x^{n-1/2} p \\ \phi_z^n &= b_z \phi_z^{n-1} + (b_z - 1)\, \partial_z^{n-1/2} p \end{aligned}\right\} \quad \text{CPML} \tag{16}$$

Donde $b_x = e^{-d(x)\Delta t}$ y $b_z = e^{-d(z)\Delta t}$. El parametro de atenuacion tomado de [10] esta dado por:

$$\left.\begin{aligned} d\left(u\right) &= d_0 \left(\frac{u}{L}\right)^2 \\ d\left(0\right) &= -\frac{3V}{2L} \ln\left(R\right) \end{aligned}\right\} \quad \text{Factor Atenuación} \tag{17}$$

Una visualización del frente de onda en las fronteras usando CPML se ve en la figura 14



(a)

(b)

(c)

Figura 14: Dinámica del frente de onda en un medio de velocidad constante aplicando condiciones de frontera CPML. Figura 14a Frente de onda 14b Frente de onda llegando a la frontera. Figura 14c Frente de onda reflejado.

3. IMPLEMENTACIÓN DE RTM 2D

Conociendo los cuidados que se deben tener en el manejo de los datos, y cómo se desarrolla la ecuación de onda en la GPU, se procede a implementar el algoritmo de RTM 2D. El código se divide en 4 subrutinas. La primera

de ellas consiste en leer el archivo de las trazas sísmicas y el campo de velocidades. La segunda de ellas calcula los parámetros de estabilidad, e inicializa las variables a utilizar, reserva los respectivos espacios de memoria en el *host* y en el *device*, y transfiere los datos necesarios por la GPU para ejecutar el *kernel*. La tercera de ellas, por medio de una bandera, soluciona el campo de la fuente y el campo de las receptores. Y por último la subrutina que aplica la condición de imagen. La ejecución del código se podrá realizar en cualquier computador de escritorio o portátil que cumpla las siguientes características:

- Sistema Operativo linux
- GPU Nvidia
- Debe tener instalado el software *Seismic Unix*
- Drivers y SDK de CUDA versión 5.5 o posterior
- Recursos de memoria RAM y disco duro necesarios.

Sin embargo, el código también podrá ejecutarse en equipo que no tengan GPU Nvidia, y este se compilará y ejecutará sobre la CPU del equipo, para ello el usuario deberá pasar como parámetro de entrada del código *'gpu=0'*, ofreciendo cierta flexibilidad en la plataforma en donde se ejecuta el programa.

El esquema usado en el manejo de los datos para la solución de la ecuación de onda, incluyendo las fronteras CPML es el presentado en la figura 15.

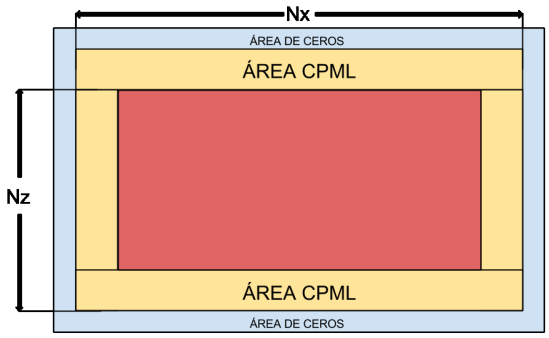


Figura 15: Esquema del cálculo del campo implementado para RTM-2D

3.1. ESTRATEGIAS DE EJECUCIÓN

Se plantean dos estrategias de ejecución. La primera de ellas, es calcular los campos y guardar ambos resultados en el disco. Luego leer el campo propagado y retropropagado del disco y aplicar la condición de imagen. Esta estrategia da como resultado 3 archivos, dos de ellos son las animaciones del campo propagado y el retropropagado y el otro es el mapa de reflectividad. Esta estrategia puede ser seleccionada para cualquiera de las ejecuciones en CPU o GPU usando la bandera *fullmemory=0*.

La segunda estrategia, es evitar la escritura en disco, y para ello el cálculo de los campos propagado y retropropagado se guardaran en memoria RAM, y luego se aplica la condición de imagen. Esta estrategia da como resultado solo el archivo del mapa de reflectividad. Esta estrategia puede ser seleccionada nuevamente para cualquiera de las ejecuciones en CPU o GPU usando la bandera *fullmemory=1*.

3.2. ANÁLISIS DE DEPENDENCIA DE DATOS

3.2.1 Dependencia de datos dados por los parámetros del usuario

La implementación de RTM-2D aprovecha las utilidades brindadas por el software *Seismic Unix*, y por ello el ingreso de parámetros se condicionó a ello.

Los parámetros que deben ser ingresados por el usuario son los siguientes:

- Ubicación del archivo de velocidades.
- Ubicación del archivo de las trazas.
- Tamaño del modelo de velocidades (Nx, Nz).
- Parámetro muestreo del modelo (dx, dz).
- Número de disparos a migrar.
- Frecuencia de la ondícula.
- Número de puntos por longitud de onda.

Como parámetros opcionales aunque el código se ejecuta con unos valores por defecto, para el CPML se podrán ingresar el número de puntos, la velocidad y el coeficiente de reflexibidad. Si el usuario no cuenta con GPU, podrá ejecutar el código en CPU, Además de si desea utilizar la estrategia de *full memory*. Para esta estrategia, el equipo en donde se ejecute deberá contar con los requerimientos en memoria necesarios.

Un diagrama sobre la dependencia inicial de los datos proporcionados por el usuario se aprecia en la figura 16.

Un pseudocódigo del programa es presentado en el algoritmo 4.

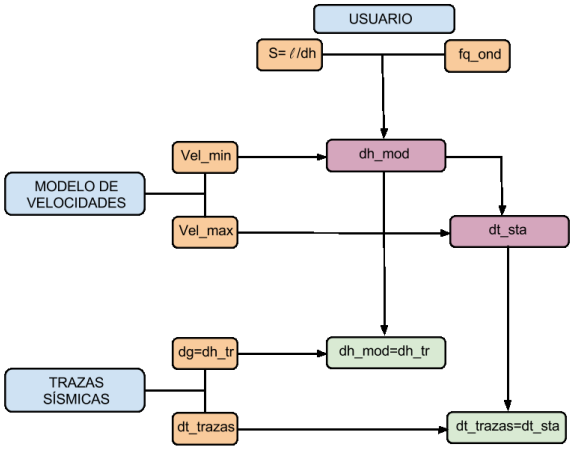


Figura 16: Diagrama de dependencia de datos de los parámetros de RTM

Data: Archivo de Trazas, Modelo de Velocidades, otros parámetros
Result: Reflective-Map.bin
Inicialización;
Funcion ReadShot;
Funcion Stability;
for *i=0; número de disparos* **do**
  if *ReadShot=null* **then**
    No se encontraron mas shots
  end
  ;
  if *gpu=1* **then**
    fdtd2DGPU-campo de la fuente;
    fdtd2DGPU-campo de los receptores;
    condición de imagen GPU;
  else
    fdtd2DCPU-campo de la fuente;
    fdtd2DCPU-campo de los receptores;
    condición de imagen CPU;
  end
  if *fullmemory* **then**
    no-Disk-transfer;
  end
end
free (all);

Algoritmo 4: main RTM2D

3.2.2 Cálculo del número de operaciones

Para el cálculo del número de operaciones no se tendrán en cuenta los aportes realizados por CPML. Estos cálculos se dividen en 2. Primero se determinará el número de operaciones necesarias para calcular un punto del campo, y luego se determinará el numero de operaciones necesarias para aplicar la condición de imagen.

Para el cálculo del número de operaciones del stencil, hay que considerar que por cada punto del aporte del laplaciano, se realiza una multiplicación, y por cada dos puntos una suma. De tal manera que para el calculo de un punto para 1 solo campo se tiene:

$$\#op_{stencil} = \frac{(Nx^*Nz)^* \, (19mult + 17sum)}{k} \tag{18}$$

Ahora para el cálculo del número de operaciones a realizar con la condición de imagen, es:

$$\#op_{condImg} = (Nx^*Nz)_{mult} + (Nx^*Nz) * nt_{sum} \tag{19}$$

El numero de operaciones totales para RTM 2D es:

$$\#op_{total} = (2 * op_{stencil}) + (op_{conImg}) \tag{20}$$

4. RESULTADOS CALIDAD DE IMAGEN

La validación del mapa de reflectividad obtenido con el algoritmo para zonas geológicas de alta complejidad fue realizada por doctora Ph.D Flor Alba Vivas Mejia codirectora del proyecto.
El set de pruebas en la calidad de las imágenes esta conformado por un modelo tipo escalón, y el modelo Marmousi creado por el Instituto Francés del Petroleo. El modelo escalón se realiza pensando en el resultado que dará el algoritmo en zonas geológicas que presenten buzamientos mientras que el modelo Marmousi representa las estructuras geológicas con variaciones en las velocidades, fenómenos estructurales como fallas, intrusiones de alta velocidad y acabalgamientos.

4.1. MODELO ESCALÓN

El modelo escalón, es un modelo sintético que tiene de 4000 [m] de distancia y 2000 [m] de profundidad. La adquisición se realizo usando la función *sufdmod2pml* del software *Seismic Unix*. Se realizaron 145 disparos. El primero de ellos en 500 [m] y la distancia entre disparos es de 20[m]. La ubicación de los geófonos se realiza en todo el modelo, iniciando desde 0[m] hasta 3990[m]. El disparo numero 72, (ver figura 17), el modelo de velocidades, (ver figura 18), y la imagen migrada, (ver figura 19) son:

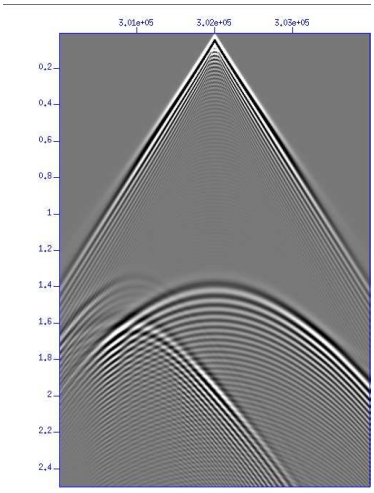


Figura 17: Trazas del disparo número 120

4.2. MRMOUSI

Marmousi es un modelo sintético creado por el IFP en 1998. Este modelo se sitúa bajo 32 [m] de agua, tiene 9.200 [m] de distancia, y 3000 [m] en profundidad. El número de disparos es de 240, el primer disparo se encuentra ubicado a 3000 [m], la distancia entre disparos es de 25 [m]. Son utilizados 96 geófonos, separados por 25 [m], y la ubicación del primero

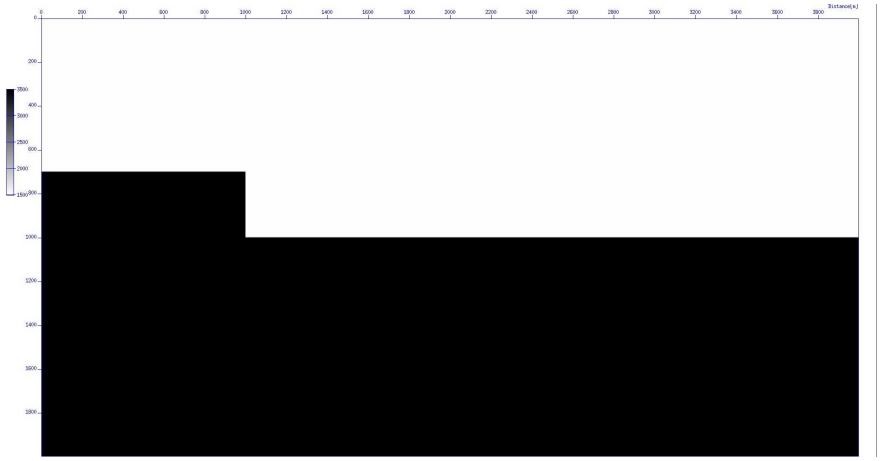


Figura 18: Modelo de velocidades escalón

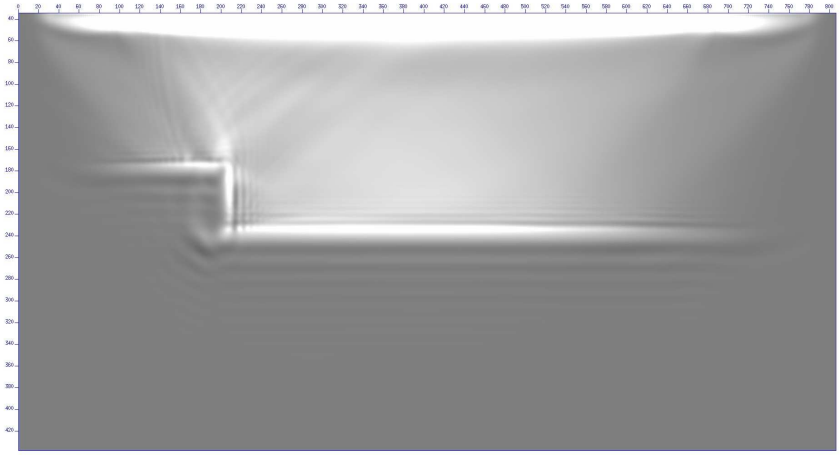


Figura 19: Imagen migrada modelo escalón

de ellos es en 425 [m].
El disparo número 140, (ver figura 20), el modelo de velocidades, (ver figura 21), y la imagen migrada, (ver figura 22) son:

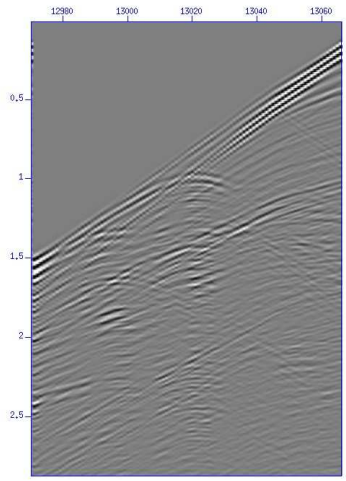


Figura 20: Disparo número 140

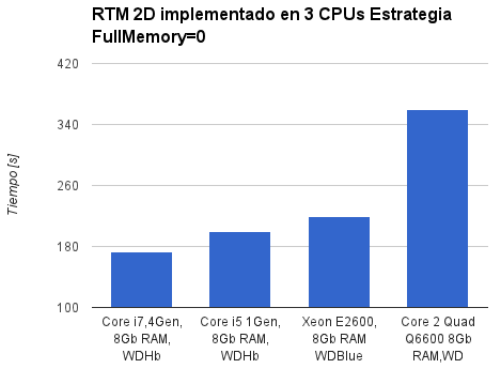


Figura 23: RTM2D-código secuencial

5. ANÁLISIS DE RENDIMIENTO

Para el análisis de rendimiento de la aplicación RTM2D, se realizaron varias pruebas para diferentes arquitecturas de computo. Ésta implementación se realizó siempre en la GPU EVGA GTX - 580 cuando ésta era seleccionada. El modelo a utilizar es Marmousi que tiene dimensiones de (1845x600), un *dh=5 [m]*, y un número de pasos de tiempo nt=5752, con un tiempo de muestreo de *dt=0.792 [ms]*. Las pruebas se realizaron migrando 1 disparo.

5.1. PRUEBA 1

Para esta prueba se evaluó el desempeño del algoritmo en diferentes arquitecturas de CPU. El código es ejecutado con las banderas *gpu=0 , fullmemory=0,* por lo tanto se busca evaluar los tiempos al momento de escritura y lectura para el cálculo de los campos y la condición de imagen.

La variación en los tiempos, se debe a la cantidad de memoria cache presente en cada CPU. Sin embargo este no es el único factor influyente para esta prueba, ya que el tipo de disco duro presente en el equipo Xeon E5-2609 es un WD-Blue 64MB de cache, y el disco usado en los otros 3 equipo es de clase híbrida. (NAND 4GB + HDD).

5.2. PRUEBA 2

En esta prueba se presenta la diferencia en tiempos de la implementación de CPU y GPU usando la estrategia *fullmemory=0*. Para esta prueba son utilizados las 2 mejores configuraciones de la anterior prueba. Esta prueba evaluara la diferencia en el desempeño de ambas implementaciones, en diferentes equipos de computo

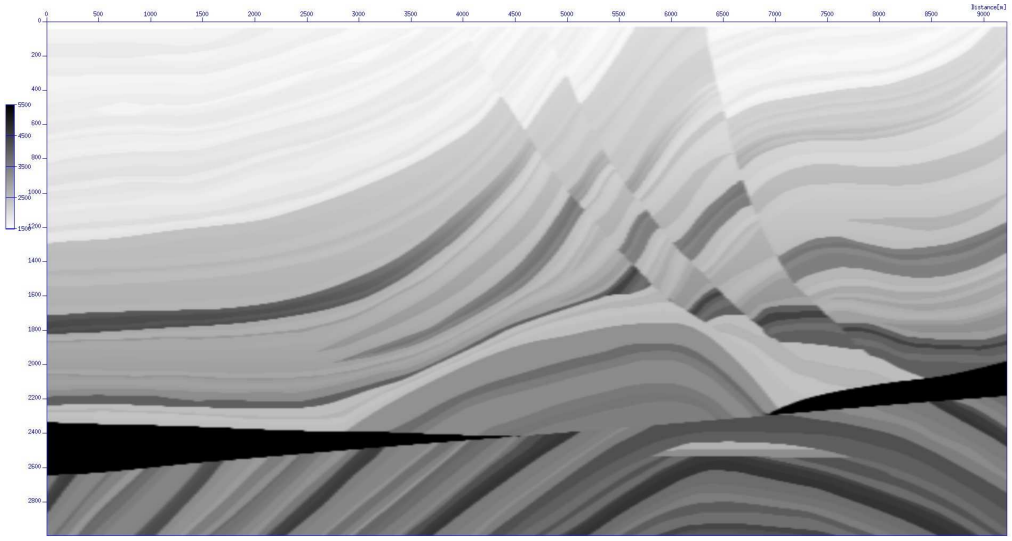


Figura 21: Modelo de velocidades Marmousi

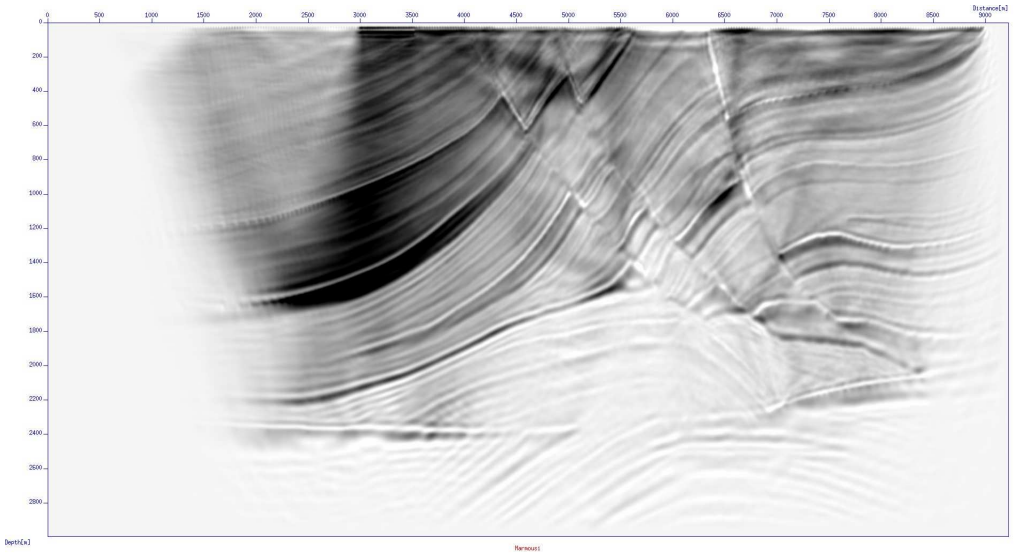


Figura 22: Imagen migrada modelo Marmousi

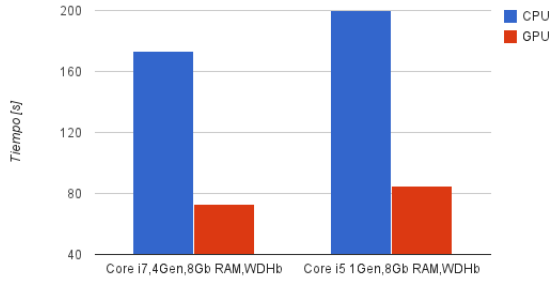


Figura 24: Comparación del tiempo en la implementación GPU-CPU

El *speedup*[13] de esta prueba para el equipo Core i7 es *speedup=2.36*, y para el equipo Core i5 es *speedup=2.24*.

5.3. PRUEBA 3

En la siguiente prueba, se busca evaluar el desempeño del algoritmo, al momento de ejecutar el código en la GPU, pero variando la estrategia de *fullmemory*. Para este caso solo 2 arquitecturas fueron usadas. Esta prueba evaluará los tiempos en la transferencia *host-disk to GPU* y *host-ram to GPU*, ademas del *speedup* en estas implementaciones.

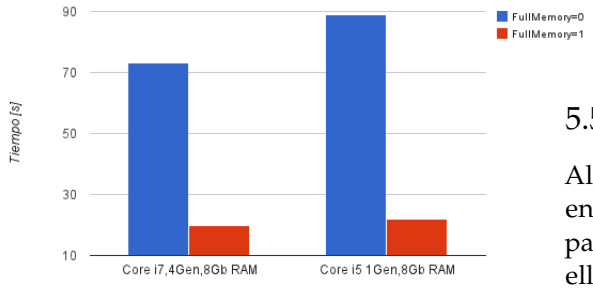


Figura 25: Desempeño la implementación usando la GPU variando la estrategia *fullmemory*

En esta prueba, se observa la clara diferencia en el desempeño de RTM, cuando no se utiliza el disco al guardar el campo propagado y retropropagado. La diferencia en el desempeño en esta prueba es superior a la anterior. Para el equipo Core i7 es *speedup=3.66*, y para el equipo Core i5 es *speedup=4.04*.

5.4. PRUEBA 4

En esta prueba, se evalúa el desempeño de RTM 2D en la GPU, pero variando la granularidad aplicada al implementar el código. Para ello, se propone variar homogéneamente la cantidad de *Threads* existentes por *Block* hasta llegar al valor máximo soportado por la arquitectura. Además, se busca evaluar los tiempos de estas variaciones teniendo en cuenta la estrategia de ejecución *fullmemory*.

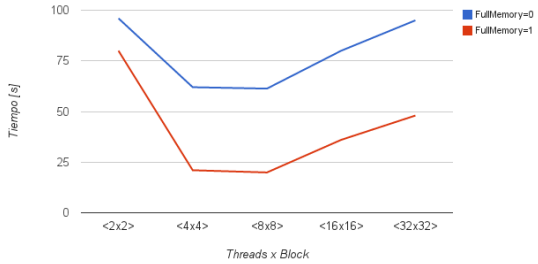


Figura 26: Desempeño la implementación variando la cantidad de *ThreadsBlock*

5.5. PRUEBA 5

Al momento de evaluar una implementación en GPU, es necesario evaluar si el uso de la capacidad de cómputo del *device* es ideal. Para ello se evalúa en los *kernels* lanzados por la aplicación, cuanto tiempo demora en la transferencia de los datos, y cuanto demora en el procesamiento. Estos datos son obtenidos usando la herramienta *nvprof* proporcionada por el SDK de CUDA.

[13] *speedup*:Cociente entre el tiempo gastado en la ejecución secuencial sobre el tiempo gastado en la ejecución en paralelo

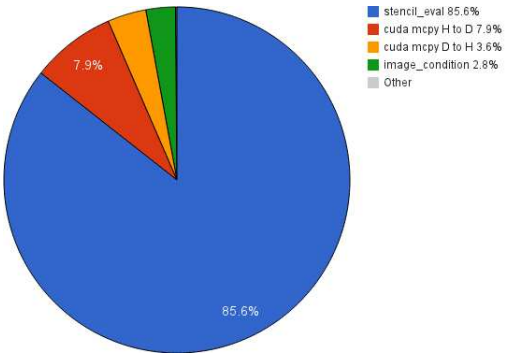


Figura 27: nvprof de los *Kernels* de RTM 2D

Como se observa, el uso de la GPU es el adecuado, debido a que los tiempos de transferencia de datos son mucho menores al tiempo que demora realizando los cálculos de los diferentes *kernels*

6. CONCLUSIONES

- Se implementó el algoritmo RTM 2D sobre una CPU y GPU analizando su código fuente y su dependencia de datos.
- Es necesario realizar un re-muestrero del modelo de velocidades usando el dh calculado para la propagación del frente de onda.
- Para obtener un mapa de reflectividad coherente fue imprescindible re-muestrar las trazas con el tiempo de muestreo que cumple la condición de estabilidad del método numérico.
- Usando la tesis de pregrado [3] como referente, la aceleración alcanzada comparando dicha implementacion con la implenetacion en CPU realizada en este proyecto es de 0.740X
- Usando la tesis de pregrado [3] como referente, la aceleración alcanzada comparando dicha implementación con la implementacion en GPU realizada en este proyecto es de 6.4X.
- La aceleración alcanzada comparando la implementación de GPU contra CPU desarrollada es de 8.7X.
- La variación en los niveles y cantidad de memoria caché al momento de ejecutar RTM 2D sobre CPU influyen en los tiempos obtenidos si los resultados son guardados en disco duro. Ver figura 23
- El tiempo de ejecución de la migración en la GPU, cuando se utiliza CPUs diferentes, usando la estrategia fullmemory=1, es similar debido a que no existe transferencia hacia el disco duro del sistema de cómputo. Ver figura 24
- Se observa una clara diferencia en los tiempos de ejecución en GPU cuando el número de *ThreadsBlock* es modificado. Ver figura 26
- El tiempo de transferencia entre el host y el device es mucho menor que el tiempo que utiliza la GPU en calcular los campos y la condición de imagen. Ver figura 27

7. OBSERVACIONES

- Al momento de desarrollar aplicaciones de cómputo intensivo, es necesario contar con un excelente equipo. Pero no simplemente es tener un presupuesto y realizar la compra de lo mejor que exista en el mercado. Antes es indispensable realizar un análisis de la aplicación y establecer las cotas mínimas para que ésta se ejecute de manera adecuada. Para este caso, la cota mínima era aportada por la cantidad de RAM y DRAM del equipo de computo. Un equipo adecuado para primeras implementaciones de RTM 2D debería contar con una cantidad mínima de RAM DE 16GB y DRAM DE 6GB. La inversión en procesadores de ultima generación con grandes cantidades de memoria L3 de cache o grandes velocidades en cada núcleo no es necesaria debido a que el procesamiento se

realizará solo en la GPU, y par obtener un mejor desempeño, todos los cálculos no deberían sobrepasar la barrera de RAM.

- En cuanto al almacenamiento del equipo, varias tecnologías están disponibles en el mercado. Se encuentran los discos duros clásicos, los híbridos y los de estado sólido (SSD). Cada uno con cualidades que se destacan respecto a los otros. Los discos duros clásicos ofrecen una capacidad de almacenamiento considerable a un precio razonable con una velocidad de lectura y escritura en promedio de 100MB/s. Los SSD ofrecen velocidades de lectura y escritura de 4 a 6 veces más rápidas que los discos clásicos, pero la capacidad de almacenamiento es menor y el costo es elevado. Además la vida útil de estos discos viene dada por un número limitado de lecturas-escrituras. Los disco híbridos manejan la flexibilidad de tener cantidades reducidas de SSD previas al almacenaje en disco magnético. Dependiendo de la aplicación a desarrollar, la elección de alguna de las 3 clases de discos resultara útil al desempeño. Si la aplicación realiza múltiples acciones de lectura y de escritura, y estos archivos son del tamaño de los GBs, la elección preferible son los discos clásicos. En cambio, si la aplicación consiste en limitadas lecturas y escrituras, y estas no alcanzan a ser del orden de los GBs, la correcta elección para este tipo de implementación serán los discos SSD.

- Respecto a las comparaciones de los tiempos de migración entre la tesis [3] y la implementación realizada en este proyecto, es necesario aclarar, que se desconoce si en [3] re-muestraron el modelo de velocidades o trabajaron con el modelo original. Ademas en [3] aplican el concepto de apertura, el cual disminuye el tiempo para la migración de cada disparo.

8. FUTURAS INVESTIGACIONES

- Dependiendo de la forma en que se realizó la adquisición sísmica, aplicar el concepto de apertura disminuiría el tiempo para la migración de cada disparo.

- Es necesario investigar sobre los parámetros de estabilidad y precisión al momento de solucionar la ecuación de onda, debido a que el comportamiento del frente de onda varía dependiendo del grosor de las capas geológicas.

- Realizar un estudio sobre el re-muestreo de las trazas, y aplicar un factor de escala es necesario para disminuir la cantidad de cálculos innecesarios sin sacrificar la calidad de la imagen en el mapa de reflectividad.

- Para futuras implementaciones sería útil realizar un modelado de la arquitectura de la GPU para predecir los tiempos de ejecución, considerando los factores lógicos y de hardware para cada aplicación.

- Es necesario un estudio detallado en la forma de implementar el *stencil* en una GPU, para aprovechar al máximo la granularidad al ejecutar RTM.

- Aprovechar las nuevas funciones en el manejo de la memoria y transferencia de datos proporcionados por la nueva versión de CUDA 6-5 *(Pinned Memory)* disminuiría el tiempo de transferencia de los datos, aumentado la aceleración de la aplicación.

- El uso de librerías de aceleración creadas por Nvidia ayudaría al mejor desempeño en la ejecución del algoritmo como AmgX, CuBlasxt y ArrayFire.

- Diferentes formas de segmentar el problema pueden ser investigadas, aprovechando que las GPUs, pueden ejecutar *kernels* concurrentes.

- El uso de múltiples GPUs para solucionar RTM aumentaría el desempeño de la aplicación aprovechando la cualidad de múltiples núcleos presentes en los procesadores actuales.
- Implementar RTM-3D usando las recomendaciones anteriores.

Agradecimientos

Primero a Dios, mi padre Nestor, mi madre Belthy, mi hermana Jenifer y mi novia Diana, por todo su apoyo y comprensión en varios momentos de mi vida. Al grupo CPS, a mi director William y al grupo RTM, y al Instituto Colombiano del Petroleo (ICP), por su gran ayuda, contribuciones técnicas y académicas para la correcta finalización de este proyecto .

BIBLIOGRAFÍA

Referencias

[1] Frederic Billette and Sverre Brandsberg-Dahl. The 2004 BP velocity benchmark., 2005.

[2] Carlos and R E Z Solano. *Doctorat ParisTech*. PhD thesis, 2013.

[3] Vivas Cataño. Analisis de la estabilidad , dispersion numérica y costo computacional de la migración RTM de la ecuacion de onda acustica en dos dimensiones, 2011.

[4] McKercher Cheng, Grossman. *Professional CUDA C Programming*. 2014.

[5] Institut Francais du P´etrole. Marmousi syntetic 2d acoustic model. `http://www.ifp.fr/IFP/en/aa.htm`. Descargado April, 2014.

[6] Bengt Fornberg. Generation of Finite Difference Formulas on Arbitrarily Spaced Grids. *Mathematics of Computation*, 51(184):699–706, October 1988.

[7] P. Bording Lines, Slawinski. A recipe for stability analysis of finite-difference wave equation computations. 10:1–6, 1998.

[8] Nvidia. Whitepaper NVIDIAs Next Generation CUDA Compute Architecture: Fermi Generation. pages 1–22.

[9] R. Bording Phillip. Finite Difference Modeling - Nearly Optimal Sponge Boundary Conditions. (October), 2004.

[10] Pengliang Yang, Jinghuai Gao, and Baoli Wang. RTM using effective boundary saving: A staggered grid GPU implementation. *Computers & Geosciences*, 68:64–72, July 2014.