EVALUACIÓN DE ESTRATEGIAS DE IMPLEMENTACIÓN DEL ALGORITMO REVERSE-TIME MIGRATION (RTM) 3D SOBRE UN NODO MULTI-GPU

IVÁN FELIPE OBREGÓN CARREÑO

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Maestría en Ingeniería Electrónica
Bucaramanga
2017

EVALUACIÓN DE ESTRATEGIAS DE IMPLEMENTACIÓN DEL ALGORITMO REVERSE-TIME MIGRATION (RTM) 3D SOBRE UN NODO MULTI-GPU

Autor:
IVÁN FELIPE OBREGÓN CARREÑO

Trabajo de investigación presentado como requisito para optar al título de
Magíster en Ingeniería Electrónica

Director:
MIE. William Alexander Salamanca Becerra

Codirector:
Ph.D. Ana Beatriz Ramírez Silva

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Maestría en Ingeniería Electrónica
Bucaramanga
2017

AGRADECIMIENTOS

Este trabajo fue desarrollado con el apoyo de ECOPETROL y COLCIENCIAS como parte del proyecto de investigación No. 0266-2013.

CONTENIDO

Pág.

INTRODUCCIÓN **11**

1 MARCO TEÓRICO **12**
1.1. ECUACIÓN DE ONDA ACÚSTICA CON DENSIDAD CONSTANTE EN DIFERENCIAS FINITAS 12
1.2. CONVOLUTIONAL PERFECT MATCHED LAYER - CPML 14
1.3. ESTÁNDARES DE COMUNICACIÓN DE CÓMPUTO EN PARALELO . 16
1.4. MESSAGE PASSING INTERFACE (MPI). 16
1.5. POSIX THREADS (PTHREADS) 17
1.6. PARALLEL VIRTUAL MACHINE (PVM) 17
1.7. SELECCIÓN DEL ESTÁNDAR DE COMUNICACIÓN 18
1.8. MIGRACIÓN SÍSMICA 20
1.9. REVERSE TIME MIGRATION 20

2 ESTRATEGIAS DE IMPLEMENTACIÓN **23**
2.1. MIGRACIÓN SÍSMICA USANDO DIVISIÓN DE DISPAROS 23
2.2. MIGRACIÓN SÍSMICA USANDO DESCOMPOSICIÓN DE DOMINIO . . 26

3 MÉTRICAS Y RESULTADOS **30**
3.1. EQUIPO DE CÓMPUTO USADO 32
3.2. RESULTADOS DE MIGRACIÓN USANDO REVERSE TIME MIGRATION 3D 33
3.3. MEDICIONES DE TIEMPO 34
3.4. ANÁLISIS DE TIEMPOS DE LA ESTRATEGIA 1 35
3.5. ANÁLISIS DE TIEMPOS DE LA ESTRATEGIA 2 36
3.6. OCUPACIÓN DE MEMORIA 37
3.7. ANÁLISIS DE OCUPACIÓN DE MEMORIA DE LA ESTRATEGIA 1 . . 38
3.8. ANÁLISIS DE OCUPACIÓN DE MEMORIA DE LA ESTRATEGIA 2 . . 39
3.9. COMPLEJIDAD COMPUTACIONAL 40

4 CONCLUSIONES **43**

REFERENCIAS **44**

BIBLIOGRAFÍA **46**

LISTA DE FIGURAS

Pág.

Figura 1. Representación gráfica del calculo del campo P futuro usando el campo presente y el pasado. 14
Figura 2. Representación de las división en dos de los recursos físicos usando MPI. 17
Figura 3. Diagrama de funcionamiento de una implementación que crea y ejecuta un proceso en paralelo en un *thread* usando *Pthreads*. 17
Figura 4. Antes y después de la migración sísmica 20
Figura 5. Esquema de funcionamiento de la primera y segunda etapa del algoritmo RTM 21
Figura 6. Esquema del funcionamiento de la aplicación de la condición de imagen. 21
Figura 7. Ejemplo de una imagen migrada usando RTM. 22
Figura 8. Diagrama de la estrategia de migración por división de disparos. . . 24
Figura 9. Funcionamiento de control de las GPUs usando ranks 24
Figura 10. Resultados parciales de la migración usando 3 GPUs. 25
Figura 11. Resultado final al sumar todas las imágenes parciales usando MPI_Reduce. 26
Figura 12. Planteamiento de la malla de diferencias finitas de octavo orden en la frontera izquierda del modelo. 27
Figura 13. Secciones compartidas entre los submodelos. 28
Figura 14. Operación de *swapping* entre dos modelos adyacente. 28
Figura 15. Operación de *swapping* entre dos modelos adyacente. Los colores indican las secciones que serán intercambiadas y las flechas la dirección del intercambio. 29
Figura 16. Modelo *SEG/EAGE Salt Model* utilizado para las pruebas. 31
Figura 17. Secciones planas del Modelo *SEG/EAGE Salt Model* utilizado para las pruebas. 31
Figura 18. Domo de sal extraído del modelo de velocidades *SEG/EAGE Salt Model*. 32
Figura 19. Esquema del cluster utilizado para la pruebas. 33
Figura 20. Secciones planas de la imagen final migrada usando RTM. 34
Figura 21. Tiempos de cómputo obtenidos por las estrategias 1 y 2. 35
Figura 22. Factor de aceleración obtenido usando la estrategia 1. 36
Figura 23. Factor de aceleración obtenido usando la estrategia 2. 37
Figura 24. Factor de reducción de memoria usando la estrategia 2. 40

LISTA DE TABLAS

Tabla 1. Tabla comparativa de las características más importantes de MPI, Pthreads y PVM. 19
Tabla 2. Detalle de los nodos utilizados. 33
Tabla 3. Norma l_2 entre las imágenes generadas usando las estrategias propuestas y la implementación de referencia. 34
Tabla 4. Tiempos medidos usando las estrategias 1 y 2. 34
Tabla 5. Medición de ocupación de memoria para la implementación de referencia y la estrategia 1. 38
Tabla 6. Medición de ocupación de memoria para la implementación de referencia y la estrategia 2. 39

RESUMEN

TÍTULO: Evaluación de estrategias de implementación del algoritmo Reverse-Time Migration (RTM) 3D sobre un nodo multi-GPU*

AUTOR: Iván Felipe Obregón Carreño**

PALABRAS CLAVE: Migración sísmica, Reverse-Time Migration, Computación de alto desempeño,

DESCRIPCIÓN:

La migración sísmica es uno de las etapas del procesamiento sísmico empleadas por la industria de los hidrocarburos para generar imágenes del subsuelo. La técnica *Reverse-Time Migration* (RTM) es uno de los métodos mas comunes ya que generan imágenes de alta calidad en escenarios con estructuras complejas. Sin embargo, este método implica un alto costo computacional porque usa la ecuación de onda completa para encontrar los campos de onda, incrementando tanto los costos computacionales como la ocupación de memoria.

En este informe, propusimos dos estrategias para la implementación del método RTM usando un nodo multi-GPU. La primera estrategia fue llamada división y consiste en dividir los datos a procesar en diferentes GPUs usando *Message Passing Interface* (MPI) con el fin de disminuir los tiempos de cómputo. La segunda estrategia es llamada descomposición de dominio y consiste en dividir la carga computacional por GPU, disminuyendo la ocupación de memoria por GPU. Se realizaron pruebas sobre las dos estrategias sobre el modelo de pruebas SEG/EAGE salt model, obteniendo un factor de aceleración en tiempos de 3,99 cuando se usan 4 GPUs y un factor de reducción de memoria hasta de 3,12, ambos resultados realizando la comparación con la implementación en una GPU.

* Trabajo de investigación.

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Maestría en Ingeniería Electrónica. Director: MIE. William Alexander Salamanca Becerra.

ABSTRACT

TITLE: Review of implementation strategies of the Reverse-time migration (RTM) 3D algorithm on a GPU cluster*

AUTHOR: Iván Felipe Obregón Carreño**

KEYWORDS: Migration, RTM, High performance Computing, Reverse-Time Migration,

DESCRIPTION:

The seismic migration is one of the seismic processing stages employed by the oil and gas industry to generate subsurface images. Reverse-Time Migration (RTM) is one of the most common methods because it generates subsurface images with high quality in scenarios with complex structures. However, the method implies a high computational cost because it uses the solution of the wave equation to find the source wavefield, increasing the runtime and the memory occupancy.

In this report, we propose two strategies to develop the RTM implementation using a GPU cluster. The first strategy is called shots division that consists in splitting the data to be processed in different GPUs using Message Passing Interface (MPI) in order to increase the speedup factor. The second strategy is named domain decomposition and consist in dividing the computational load into the GPUs in the cluster, decreasing the memory occupancy by GPU. We tested the strategies by using the synthetic SEG/EAGE salt model, obtaining a speed up factor of 3,99 when 4 GPUs are used and a memory reduction factor of 3,12 in comparison to the implementation in a single GPU.

* Trabajo de investigación.

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Maestría en Ingeniería Electrónica. Director: MIE. William Alexander Salamanca Becerra.

INTRODUCCIÓN

En la industria de los hidrocarburos, la exploración sísmica es un pilar fundamental para su funcionamiento debido a la alta demanda de petróleo[1], el cual se utiliza para satisfacer los campos de la energía, transporte y otros mercados. La exploración sísmica en la búsqueda de petróleo, gas u otros minerales está compuesta por etapas entre las que se destacan la adquisición, inversión, migración, análisis e interpretación entre otras. La migración sísmica tiene como objetivo generar la imagen final del subsuelo la cual será usada para su posterior análisis e interpretación.

Actualmente, existen diferentes técnicas de migración sísmica, entre ellas están la *One-Way Wave Equation Migration* (OWWEM) y la *Reverse Time Migration* (RTM). OWWEM es la técnica más usada debido a su bajo costo computacional y su aceptable grado de precisión. Por otra parte, RTM usa la ecuación de onda completa, la cual genera una mejor y más precisa imagen final migrada en áreas con topología compleja, pero provocando un alto costo computacional [2].

RTM fue introducido por Baysal en 1983 [3] que usa la ecuación de onda completa para modelar la propagación de onda acústica. A pesar de generar una imagen final del subsuelo mas detallada, los costos computacionales como la ocupación de memoria y los tiempos de ejecución son altos, debido a al gran número de operaciones que se realizan. Con el fin de solventar estos problemas, se han propuesto nuevos métodos que reduzcan el impacto a la hora de la implementación de la técnica RTM. La utilización de unidades de procesamiento gráfico, también llamadas GPUs, permiten ejecutar varias tareas al mismo tiempo [4] [5] [6], utilizando un paradigma de programación en paralelo, reduciendo los tiempos de procesamiento de los algoritmos.

Tradicionalmente, las GPUs son programadas en lenguaje CUDA-C y son alojadas en *clusters*, donde estas puedan tener un canal de comunicación físico entre ellas y el sistema de cómputo CPU, permitiendo realizar implementaciones en múltiples GPUs, logrando así realizar implementaciones concurrentes entre GPUs.

En este trabajo de investigación se presentan dos estrategias de implementación de la técnica RTM-3D utilizando un *cluster* de GPUs con el fin de reducir el impacto de los costos computacionales. La primera se enfoca en mitigar el problema asociado a los tiempos de cómputo llamada migración sísmica por división de disparos, la segunda estrategia se enfoca en la reducción de memoria por GPU definida como migración usando descomposición en dominio.

1. MARCO TEÓRICO

1.1. ECUACIÓN DE ONDA ACÚSTICA CON DENSIDAD CONSTANTE EN DIFERENCIAS FINITAS

Uno de los núcleos principales del algoritmo de migración *RTM* es el modelado de la propagación de onda acústica a través de un medio de velocidades acústicas. Este modelado parte de su base, la ecuación de onda completa la cual está descrita a continuación:

$$\frac{\partial^2 \mathbf{P}}{\partial t^2} = C^2 \cdot \bigtriangledown^2 \mathbf{P}, \tag{1.1}$$

expandiendo el operador laplaciano obtenemos

$$\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial t^2} = C^2(x,y,z) \cdot \left(\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial x^2} + \frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial y^2} + \frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial z^2} \right) \tag{1.2}$$

donde $\mathbf{P}(x,y,z,t)$ es el campo de presión escalar, $C(x,y,z)$ es el modelo de velocidades del subsuelo, x, y y z son las variables espaciales y t es la variable temporal.

La Ecuación 1.2 es implementada usando diferencias finitas centradas en el dominio temporal (FDTD), usando un *stencil* de segundo orden de aproximación para la derivada temporal, y un *stencil* de octavo orden de aproximación para las derivadas espaciales [7].

La aproximación de segundo orden para la derivada temporal es definida como

$$\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial t^2} \approx \frac{\mathbb{P}^{n+1}_{i,j,k} - 2 \cdot \mathbb{P}^{n}_{i,j,k} + \mathbb{P}^{n-1}_{i,j,k}}{\Delta t^2}, \tag{1.3}$$

donde $\mathbb{P}^{n}_{i,j,k}$ es el campo de presión discreto; los índices i,j,k representa las variables discretas para x, y y z respectivamente, el superíndice n denota la variable discreta para el tiempo t por lo tanto ($\mathbb{P}^{n-1}_{i,j,k}$), ($\mathbb{P}^{n}_{i,j,k}$) y ($\mathbb{P}^{n+1}_{i,j,k}$) denotan los campos pasado, presente y futuro, y Δt es el paso temporal.

La aproximación de octavo orden para cada una de las componentes x, y, y z es definida como

$$\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial x^2} \approx \frac{\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^{n}_{i+a,j,k}}{\Delta x^2}, \tag{1.4}$$

$$\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial y^2} \approx \frac{\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i,j+a,k}}{\Delta y^2}, \tag{1.5}$$

$$\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial z^2} \approx \frac{\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i,j,k+a}}{\Delta z^2}, \tag{1.6}$$

donde C_s es un vector cuyas posiciones van desde -4 hasta 4 y están definidas como $C_s = [\frac{-1}{560}, \frac{8}{315}, \frac{-1}{5}, \frac{8}{5}, \frac{-205}{72}, \frac{8}{5}, \frac{-1}{5}, \frac{8}{315}, \frac{-1}{560}]$ y representan los coeficientes de la aproximación de octavo orden de la doble derivada en diferencias finitas centradas; y Δx, Δy y Δz representan la resolución espacial del modelo.

Reemplazando las Ecuaciones 1.3, 1.4, 1.5 y 1.6 en la Ecuación 1.2 tenemos

$$\begin{aligned}\frac{\mathbb{P}^{n+1}_{i,j,k} - 2 \cdot \mathbb{P}^n_{i,j,k} + \mathbb{P}^{n-1}_{i,j,k}}{\Delta t^2} = (\mathbf{C}^n_{i,j,k})^2 \left(\frac{\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i+a,j,k}}{\Delta x^2} + \right. \\ \left. \frac{\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i,j+a,k}}{\Delta y^2} + \frac{\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i,j,k+a}}{\Delta z^2} \right) .\end{aligned} \tag{1.7}$$

Si se despeja el campo de presión futuro y tomamos que $\Delta x = \Delta y = \Delta z = \Delta h$, tenemos

$$\begin{aligned}\mathbb{P}^{n+1}_{i,j,k} =& 2 \cdot \mathbb{P}^n_{i,j,k} + \frac{(\mathbf{C}^n_{i,j,k})^2 \cdot \Delta t^2}{\Delta h^2} \left(\sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i+c,j,k} + \right. \\ & \left. \sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i,j+c,k} + \sum_{a=-4}^{4} C_s[a] \cdot \mathbb{P}^n_{i,j,k+c} \right) - \mathbb{P}^{n-1}_{i,j,k} \ ;\end{aligned} \tag{1.8}$$

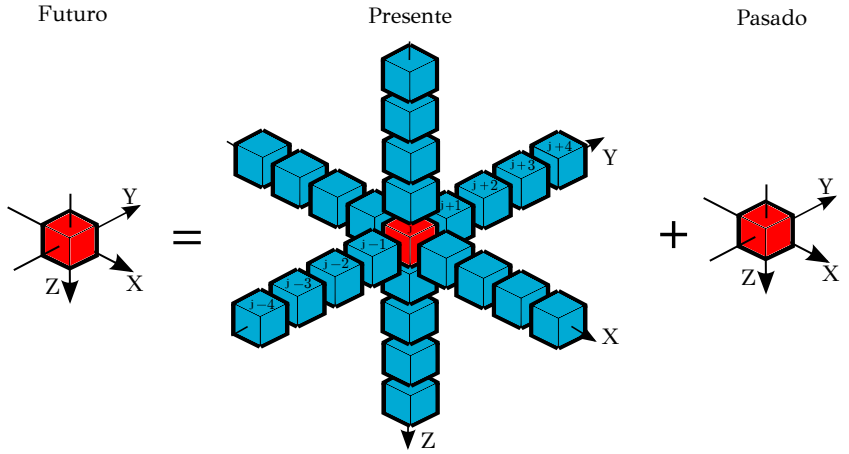
factorizando y despejando el campo futuro se obtiene

$$\mathbb{P}^{n+1}_{i,j,k} = 2 \cdot \mathbb{P}^n_{i,j,k} + \frac{(\mathbf{C}^n_{i,j,k})^2 \cdot \Delta t^2}{\Delta h^2} \left[\sum_{c=-4}^{4} C_s[c] \cdot \left(\mathbb{P}^n_{i+c,j,k} + \mathbb{P}^n_{i,j+c,k} + \mathbb{P}^n_{i,j,k+c} \right) \right] - \mathbb{P}^{n-1}_{i,j,k} \ , \tag{1.9}$$

La Ecuación 1.9 nos muestra que el campo de presión futuro $(n+1)$ se construye de manera iterativa, teniendo como valores iniciales los valores de los campos presente (n) y pasado $(n-1)$, también se observa que el tamaño de la malla de diferencias finitas (*stencil*) es de 4 para cada uno de sus direcciones $(\pm i, \pm j, \pm k)$, por lo tanto es necesario plantear unas condiciones de fronteras para que la implementación sea estable.

El comportamiento dinámico de la implementación de la Ecuación 1.9 se muestra en la Figura 1, donde se puede observar que para calcular un punto del campo futuro $\mathbf{P}$ es necesario la información espacial de 25 puntos del campo de onda presente y un punto del campo onda pasado, los cuales deben ser inicializados o almacenados para próximas iteraciones.

Figura 1: Representación gráfica del calculo del campo P futuro usando el campo presente y el pasado.



1.2. CONVOLUTIONAL PERFECT MATCHED LAYER - CPML

En un medio continuo, tal y como es la naturaleza donde no existen fronteras, la propagación de las ondas se transmiten hasta el infinito ya que no existe ningún límite que las detenga sin contar las atenuaciones del medio. Esta característica continua del medio hay que acotarla a la hora de realizar un correcto modelado de la onda acústica, y si no se tuviera en cuenta la propagación de la onda acústica sobre el modelo finito ocasionaría reflexiones no reales, afectando el fenómeno físico. Por esta razón es necesario implementar una condición de frontera para atenuar estas reflexiones no deseadas sobre el modelo que se esté trabajando. En la literatura se han propuesto varios métodos para implementar estas condiciones entre ellos se destacan *Convolutional Perfectly Matched Layer* (CPML) [8].

La ecuación de onda acústica sobre un medio isótropo con densidad constante planteada en 1.2, según [8] queda descrita así

$$\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial t^2} = C^2(x,y,z)\cdot\left(\frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial x^2} + \frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial y^2} + \frac{\partial^2 \mathbf{P}(x,y,z,t)}{\partial z^2} + \psi(x,y,z) + \zeta(x,y,z)\right), \tag{1.10}$$

donde

$$\psi(x,y,z) = \frac{\partial \psi_x(x,y,z)}{\partial x} + \frac{\partial \psi_y(x,y,z)}{\partial y} + \frac{\partial \psi_z(x,y,z)}{\partial z} \tag{1.11}$$

y

$$\zeta(x,y,z) = \zeta_x(x,y,z) + \zeta_y(x,y,z) + \zeta_z(x,y,z). \tag{1.12}$$

Se define ψ y ζ como variables auxiliares para la atenuación en las fronteras y se describen mediante las siguientes ecuaciones:

$$\psi_q = b_q \psi_q + b_q \left(\frac{\partial \mathbf{P}}{\partial q} \right), \tag{1.13}$$

$$\zeta_q = b_q \zeta_q + a_q \left(\frac{\partial^2 \mathbf{P}}{\partial q^2} + \frac{\partial \psi_q}{\partial q} \right) \tag{1.14}$$

donde ($q \in \{x, y, z\}$) y los coeficientes b_q y a_q determinan la atenuación que generan las variables auxiliares y son calculados a partir de la zona del modelo, la frecuencia de la fuente y la velocidad del medio. Y son calculados de la siguiente manera:

$$b_q = e^{-(\sigma_q + \alpha_q)dt}, \tag{1.15}$$

$$a_q = \frac{\sigma_q}{\sigma_q + \alpha_q}(b_q - 1), \tag{1.16}$$

donde σ es el factor de amortiguamiento el cual controla la absorción de la onda acústica y α es un parámetro del sistema que ocasiona el corrimiento del polo al origen [8]. Por ende, la ecuacion 1.10 en diferencias finitas se define de la siguiente manera:

$$\psi_q : \begin{cases} \mathbf{\Psi}_{\mathbf{x}(i,j,k)} = b_{x(i)} \mathbf{\Psi}_{\mathbf{x}(i,j,k)} + a_{x(i)} \sum_{c=-4}^{4} \left\{ \frac{1}{\Delta h} C_f[c] \cdot \mathbb{P}^n_{i+c,j,k} \right\}, \\ \mathbf{\Psi}_{\mathbf{y}(i,j,k)} = b_{y(j)} \mathbf{\Psi}_{\mathbf{y}(i,j,k)} + a_{y(j)} \sum_{c=-4}^{4} \left\{ \frac{1}{\Delta h} C_f[c] \cdot \mathbb{P}^n_{i,j+c,k} \right\}, \\ \mathbf{\Psi}_{\mathbf{z}(i,j,k)} = b_{z(k)} \mathbf{\Psi}_{\mathbf{z}(i,j,k)} + a_{z(k)} \sum_{c=-4}^{4} \left\{ \frac{1}{\Delta h} C_f[c] \cdot \mathbb{P}^n_{i,j,k+c} \right\}, \end{cases} \tag{1.17}$$

$$\zeta_q : \begin{cases} \mathbf{Z}_{\mathbf{x}(i,j,k)} = b_{x(i)} \mathbf{Z}_{\mathbf{x}(i,j,k)} + a_{x(i)} \sum_{c=-4}^{4} \left\{ \frac{1}{\Delta h^2} C_s[c] \cdot \mathbb{P}^n_{i+c,j,k} + \frac{1}{\Delta h} C_f[c] \cdot \mathbf{\Psi}_{\mathbf{x}(i+c,j,k)} \right\}, \\ \mathbf{Z}_{\mathbf{y}(i,j,k)} = b_{y(j)} \mathbf{Z}_{\mathbf{y}(i,j,k)} + a_{y(i)} \sum_{c=-4}^{4} \left\{ \frac{1}{\Delta h^2} C_s[c] \cdot \mathbb{P}^n_{i,j+c,k} + \frac{1}{\Delta h} C_f[c] \cdot \mathbf{\Psi}_{\mathbf{y}(i,j+c,k)} \right\}, \\ \mathbf{Z}_{\mathbf{z}(i,j,k)} = b_{z(k)} \mathbf{Z}_{\mathbf{z}(i,j,k)} + a_{z(i)} \sum_{c=-4}^{4} \left\{ \frac{1}{\Delta h^2} C_s[c] \cdot \mathbb{P}^n_{i,j,k+c} + \frac{1}{\Delta h} C_f[c] \cdot \mathbf{\Psi}_{\mathbf{z}(i,j,k+c)} \right\}, \end{cases} \tag{1.18}$$

$$\begin{aligned} \mathbb{P}^{n+1}_{i,j,k} =& 2 \cdot \mathbb{P}^n_{i,j,k} + \frac{(\mathbf{C}^n_{i,j,k} \cdot \Delta t)^2}{\Delta h^2} \left[\sum_{c=-4}^{4} C_s[c] \cdot \left(\mathbb{P}^n_{i+c,j,k} + \mathbb{P}^n_{i,j+c,k} + \mathbb{P}^n_{i,j,k+c} \right) \right] \\ &+ \frac{(\mathbf{C}^n_{i,j,k} \cdot \Delta t)^2}{\Delta h} \left[\sum_{c=-4}^{4} C_f[c] \cdot \left(\mathbf{\Psi}_{\mathbf{x}(i+c,j,k)} + \mathbf{\Psi}_{\mathbf{y}(i,j+c,k)} + \mathbf{\Psi}_{\mathbf{x}(i,j,k+c)} \right) \right] \\ &+ (\mathbf{C}^n_{i,j,k} \cdot \Delta t)^2 \left[\mathbf{Z}_{\mathbf{x}(i,j,k)} + \mathbf{Z}_{\mathbf{y}(i,j,k)} + \mathbf{Z}_{\mathbf{z}(i,j,k)} \right] - \mathbb{P}^{n-1}_{i,j,k} \end{aligned} \tag{1.19}$$

donde $\mathbf{\Psi}_{\mathbf{q}(i,j,k)}$, $\mathbf{Z}_{\mathbf{q}(i,j,k)}$ son los volúmenes discretos de ψ_q y ζ_q. La variable C_f representa los coeficientes de la aproximación de octavo orden en diferencias finitas centradas para la derivada de primer orden, $C_f = [\frac{1}{280}, \frac{-4}{105}, \frac{1}{5}, \frac{-4}{5}, 0, \frac{4}{5}, \frac{-1}{5}, \frac{4}{105}, \frac{-1}{280}]$; Las posiciones del vector C_f van desde -4 hasta 4.

1.3. ESTÁNDARES DE COMUNICACIÓN DE CÓMPUTO EN PARALELO

En informática, un estándar de comunicación es un sistema de reglas o directrices que permiten a dos o más entidades transmitir o recibir cualquier tipo de dato digital. El estándar también define la sintaxis, la sincronización y los métodos de recuperación de errores. Existen diversos estándares de comunicación los cuales permiten realizar comunicación entre dispositivos ya sea CPU o GPU, entre estos se encuentran MPI, Pthreads y PVM entre otros. Estos estándares poseen características en común tales como el poder paralelizar programas ya sean usando el modelo SPMD[1] (*Single Program Multiple Data*) o MPMD[2] (*Multiple Program Multiple Data*).

1.4. MESSAGE PASSING INTERFACE (MPI).

El estándar MPI fue lanzado en 1994 y desarrollado de manera comunitaria por más de 40 organizaciones y cuyo objetivo era crear una librería de paso de mensajes que permitiera crear programas que pudieran ser migrados a diferentes computadores conectados en paralelo [9].

La principal característica de MPI es manejar los modelos SPMD y MPMD, lo cual permite al usuario escribir programas como un proceso secuencial del que se podrán lanzar en múltiples procesadores. Estos procesos invocan diferentes tareas tales como:

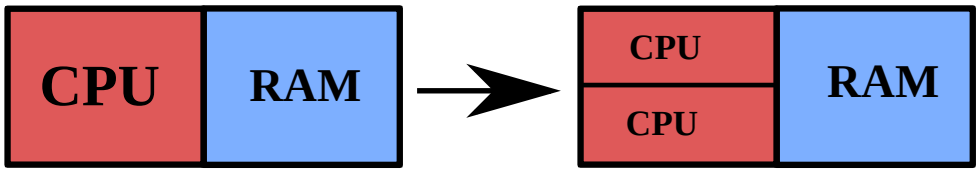
- Iniciar, gestionar y finalizar procesos MPI.
- Comunicar datos entre procesos o grupos de procesos.
- Distribuir la memoria entre los procesos MPI.

MPI funciona usando un modelo de memoria distribuida, es decir, gestiona los recursos CPU para crear procesos independientes compartiendo los recursos de memoria física, pero teniendo su propia organización de punteros. El diagrama e de la distribución recursos físicos de una implementación MPI se muestra en 2.

[1] Es una técnica empleada para lograr paralelismo, y su funcionamiento consiste en tomar una tarea para ser ejecutadas en múltiples procesadores y obtener resultados más rápidos.

[2] Se basa en tomar un serie de tareas y dividirlas para que sean procesadas en diferentes procesadores de manera independiente.

Figura 2: Representación de las división en dos de los recursos físicos usando MPI.

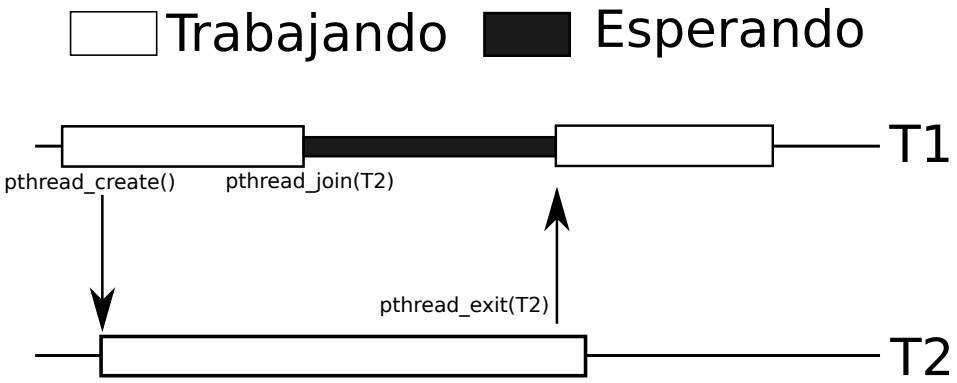


1.5. POSIX THREADS (PTHREADS)

Pthreads es un modelo estándar para la división de programas en sub-tareas cuya ejecución puede ser intercalada u organizada por lanzamientos de programa de manera paralela. El origen del nombre viene del *Portable Operating System* que significa la letra "P" y su complemento *threads* la cual significa hilos computacionales [10].

Este modelo usa la técnica de *Multithreading*, la cual permite que un programa pueda realizar múltiples tareas de manera simultánea. Su funcionamiento se basa en la creación de funciones o tareas las cuales solo uno o un grupo de *threads* trabajaran, teniendo como entradas a cada uno de ellos los atributos característicos para su tarea [11]. El diagrama de funcionamiento de un programa que crea y ejecuta un proceso en paralelo usando *Pthreads* se ve en la Figura 3.

Figura 3: Diagrama de funcionamiento de una implementación que crea y ejecuta un proceso en paralelo en un *thread* usando *Pthreads*.



1.6. PARALLEL VIRTUAL MACHINE (PVM)

PVM es un sistema para el trabajo entre redes interconectadas heterogéneas que tiene como objetivo general crear soluciones a la computación concurrente. PVM se enfoca en explotar al máximo el hardware del equipo de cómputo sin muchos gastos adicionales[12].

El sistema de PVM está compuesto por dos partes. La primera se describe como un *daemon*[3] el cual es el encargado de crear la maquina virtual donde serán ejecutados los procesos y que podrán ser iniciados desde cualquier *host*. La segunda parte son las librerías del sistema las cuales contiene todo el repertorio de primitivas que se necesitan para realizar co-procesamiento.

1.7. SELECCIÓN DEL ESTÁNDAR DE COMUNICACIÓN

En el estado del arte existen diferentes estándares que permiten la paralelización y comunicación entre dispositivos. Para realizar la selección, se creó una tabla comparativa entre los 3 estándares para conocer sus ventajas y sus desventajas y así poder escoger el más conveniente para el proyecto. Los estándares que se compararon fueron MPI, Pthreads y PVM. Las comparaciones se pueden ver en la tabla 1.

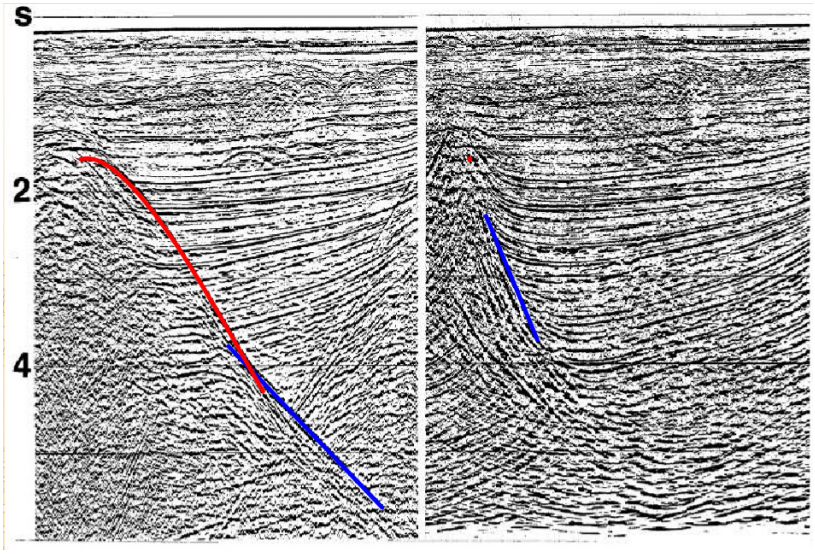
En la tabla 1 se puede detallar que los tres manejan los mismos tipos de modelo de paralelización y que el control de procesos se realiza de manera dinámica usando PVM, independiente usando Pthreads o de manera comunitaria usando señales en el caso de MPI, pero la característica más importante para este proyecto es la comunicación y en esta destaca MPI, ya que al poseer una topología virtual y una comunicación robusta hace que el envío de mensajes entre procesos, o ya aplicado al proyecto, entre GPUs sea mas sencilla y optimizada, es por esta razón que se escogió MPI como el estándar de comunicación para este proyecto. Además de las razones anteriores, la empresa distribuidora de las GPUs utilizadas (NVIDIA) recomienda el estándar MPI para realizar implementaciones concurrentes en *clusters* conformadas por sus GPUs[13].

[3] Es un proceso informático que se ejecuta en segundo plano.

Tabla 1: Tabla comparativa de las características más importantes de MPI, Pthreads y PVM.

Caracteristica	MPI	Pthreads	PVM
Modelo Utilizado	SPMD y MPMD.	SPMD y MPMD.	SPMD y MPMD.
Control de procesos	Se puede enviar señales de control para iniciar o finalizar procesos.	Cada hilo posee su propio flujo de control hasta que el hilo principal finalize.	Puede crear o eliminar procesos de manera dinámica.
Topología Virtual	Puede nombrar los procesos de manera conveniente para crear patrones y así optimizar la comunicación entre ellos.	No posee.	No posee.
Comunicación	Tiene un soporte robusto en la comunicación entre procesos.	Su comunicación es simple y se realiza enviando señales de activación.	Su comunicación es simple y se controla por el hilo principal.

Figura 4: Antes y después de la migración sísmica. Adaptado de [14]



1.8. MIGRACIÓN SÍSMICA

La migración sísmica es una herramienta importante en el procesamiento de datos adquiridos en campo y cuyo objetivo es reubicar los eventos de profundidad del subsuelo a su verdadera posición y colapsar las difracciones del mismo, produciendo un incremento en la resolución espacial generando así una imagen sísmica del subsuelo [14]. En la figura 4 se observa el fenómeno de migración, reubicando el reflector azul a su verdadera posición y colapsando la difracción roja (hipérbola) a su punto de dispersión.

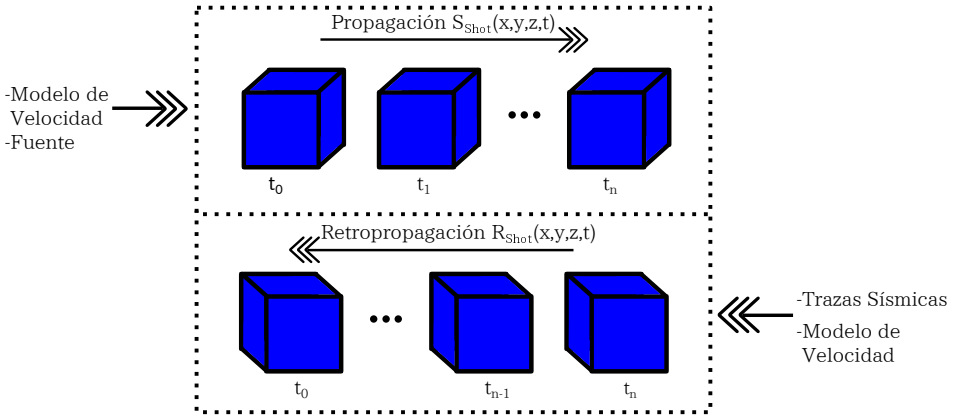
1.9. REVERSE TIME MIGRATION

RTM es una técnica de migración que usa como núcleo principal la ecuación de onda completa descrita en la ecuación (ecuación 1.2), eso quiere decir que no realiza ninguna aproximación matemática, obteniendo así una mejor y más acertada imagen migrada (imagen del subsuelo), pero conllevando un alto costo computacional [2]. El costo computacional radica en los tiempos de cómputo y en la memoria necesaria para realizar este algoritmo.

El algoritmo RTM consiste en 3 pasos principales: la propagación hacia adelante (propagación), la propagación hacia atrás (retropropagación) y la aplicación de la condición de imagen. En la figura en 5 se observa la primera y segunda etapa del algoritmo. La primera etapa usa como datos de entrada el modelo de velocidades y una fuente de excitación para así calcular el campo de onda de las fuentes (S_{Shot}), desde el tiempo inicial t_0 hasta t_n. En la segunda etapa, la retropropagación se realiza de manera similar teniendo como entradas el mismo modelo de velocidad y las trazas sísmicas obtenidas en

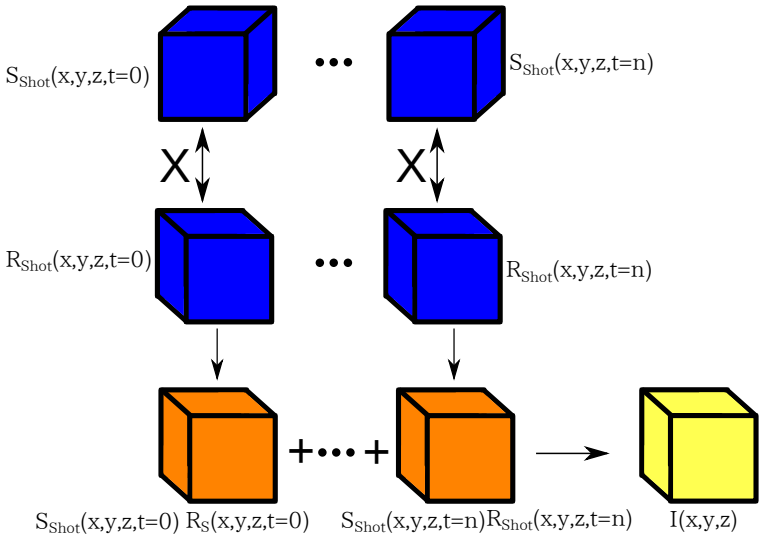
la adquisición, empezando desde el tiempo final t_n hasta el tiempo inicial t_0 obteniendo el campo de onda de los receptores (R_{Shot}).

Figura 5: Esquema de funcionamiento de la primera y segunda etapa del algoritmo RTM.



La figura 6 se observa el esquema de funcionamiento de la tercera etapa del algoritmo. La aplicación de la condición de imagen se realiza tomando los dos campos calculados en las etapas anteriores para luego ser multiplicados cada uno en sus respectivos tiempos, y posterior a esto se suman punto a punto y se reduce el campo a las dimensiones del modelo de velocidades, este resultado es la imagen migrada para ese disparo, este procedimiento se repite para todos los disparos que se hayan realizado en la adquisición. En la figura 7 se observa el resultado de una migración 2D usando RTM.

Figura 6: Esquema del funcionamiento de la aplicación de la condición de imagen.



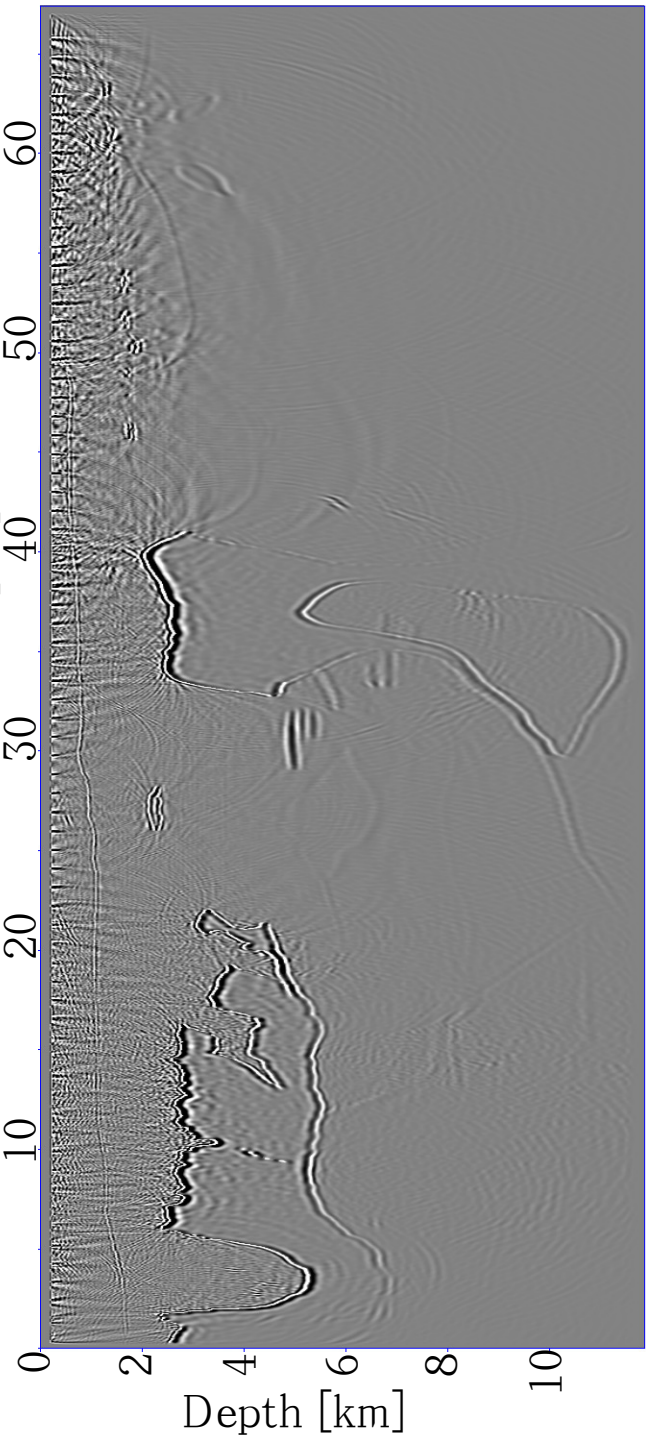


Figura 7: Ejemplo de una imagen migrada usando RTM.

2. ESTRATEGIAS DE IMPLEMENTACIÓN

Para este trabajo de investigación se implementaron dos estrategias, cada una de ellas enfocada en 2 aspectos críticos del algoritmo RTM. La primera estrategia busca mejorar los tiempos de cómputo del algoritmo, ya que estos son muy altos debido a la necesidad de calcular 2 campos de onda para cada disparo que se haya realizado [15]. Esta estrategia se denominó migración RTM usando división de disparos, la cual se basa en aprovechar la independencia de los disparos para poder dividir la carga computacional entre varias GPUs. Por otra parte, la segunda estrategia esta enfocada hacia el problema de memoria, específicamente la RAM de las GPUs. La estrategia que se planteó se denomina migración sísmica usando descomposición de dominio y su funcionamiento consiste en dividir el modelo de velocidades entre el número de GPUs que se posean, reduciendo el tamaño de los modelos de velocidades y por ende el tamaño de sus respectivos campos de onda por GPU.

2.1. MIGRACIÓN SÍSMICA USANDO DIVISIÓN DE DISPAROS

Esta estrategia tiene como objetivo reducir los tiempos de cómputo del algoritmo RTM aprovechando la independencia de los disparos para que puedan ser procesados y migrados de manera independiente. En la Figura 8 se puede observar el esquema de funcionamiento de la estrategia. Primero se cargan todos los disparos para posteriormente ser divididos en aproximadamente partes iguales dependiendo del número de GPUs que se vayan a usar. Para poder hacer esta división y que cada GPU posea su fracción de dato de los shots se usó MPI. MPI divide los recursos CPU entre el número de GPUs que se vayan a utilizar mas 1, ya que existirá un nodo maestro. Estas divisiones serán llamados *ranks*[4] y de esta manera podremos controlar cada GPU de manera independiente. En la figura 9 se observa la interacción entre los *ranks* y las GPUs.

Una vez realizada la división de los disparos, se procede a aplicar la migración RTM generando una imagen migrada para cada GPU utilizada, para así generar las imágenes parciales. En la figura 10 se ve un ejemplo $2D$ de la estrategia usando 3 GPUs. Una vez generada las imágenes parciales se utiliza una instrucción de MPI llamada MPI_Reduce la cual se encargará de sumar todas las imágenes parciales y así generar la imagen final migrada, como se ve en la figura 11.

[4] Id del proceso que fue dividido o clonado para que sea ejecutado de manera paralela.

Figura 8: Diagrama de la estrategia de migración por división de disparos.

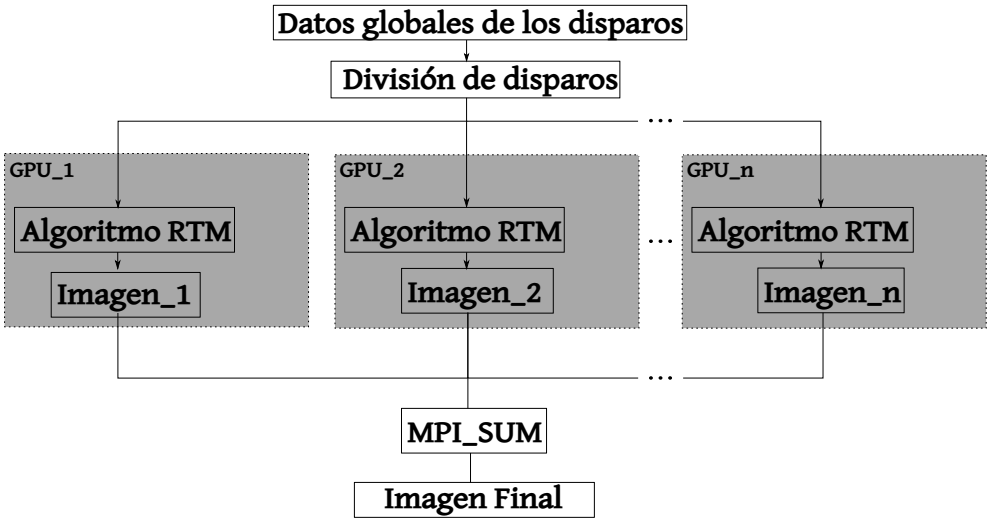


Figura 9: Funcionamiento de control de las GPUs usando ranks. El área sombreada denota las GPU que controlará cada rank, el rank 0 se encargará de coordinar todos los ranks.

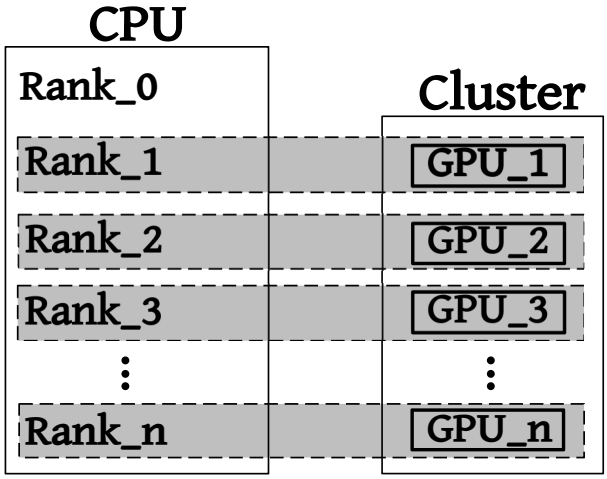


Figura 10: Resultados parciales de la migración usando 3 GPUs.

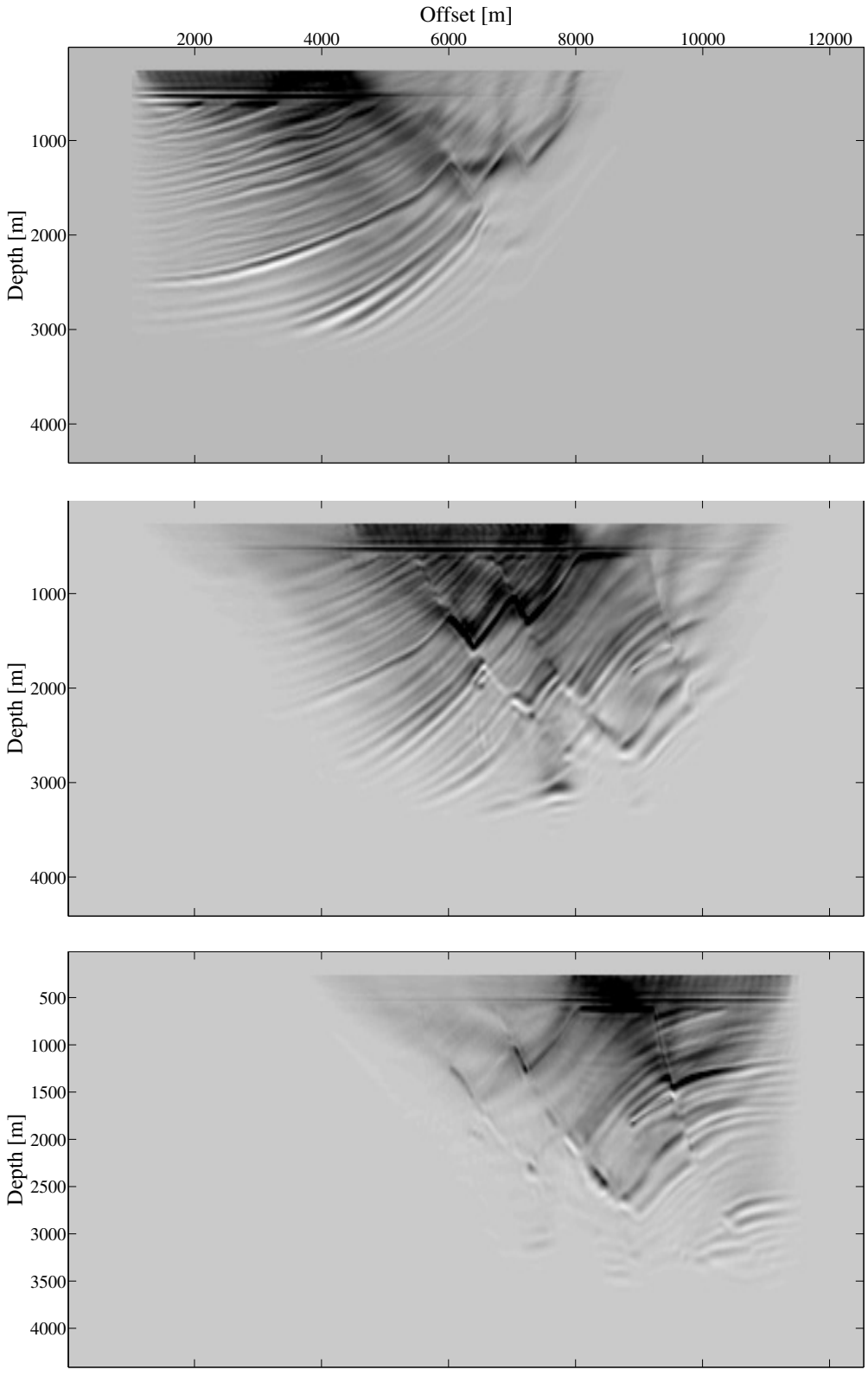
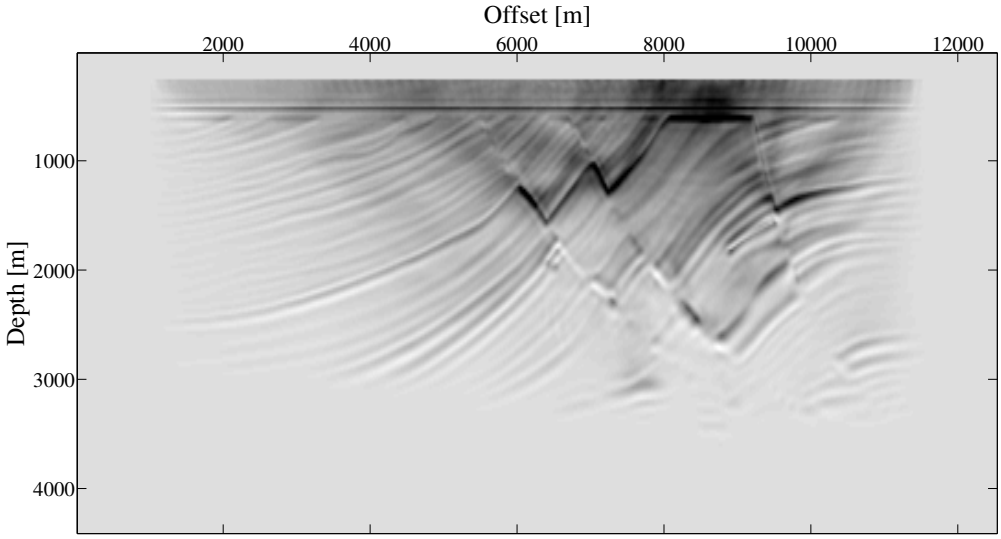


Figura 11: Resultado final al sumar todas las imágenes parciales usando MPI_Reduce.



2.2. MIGRACIÓN SÍSMICA USANDO DESCOMPOSICIÓN DE DOMINIO

Esta segunda estrategia tiene como objetivo reducir el tamaño de la memoria usada por la GPU. La descomposición de dominio consiste en asignar subdominios a partir de una malla computacional completa, con el fin de separarlos y ser procesados de manera simultánea. Uno de lo mejores ejemplos para realizar una descomposición de dominio son las implementaciones en diferencias finitas uniformes [16]. La manera como se lleva a cabo la descomposición de dominio sobre la migración RTM es aplicándola sobre el módulo de propagación de onda, ya que éste es el núcleo principal del algoritmo.

Para explicar el funcionamiento de la estrategia se tomará como ejemplo la propagación de onda acústica 2D con densidad constante, ya que el caso 3D es una extrapolación de estos mismos conceptos. Para empezar se plantea la malla de aproximación por diferencias finitas de octavo orden así como se ve en la Figura 12. En esta figura se puede detallar el problema de las condiciones de frontera que existe al usar diferencias finitas, ya que dependiendo del orden de la aproximación, los valores de las orillas (color naranja) no podrán ser calculadas y los únicos valores que si podrán ser calculados serán aquellos a partir de la columna 5 en la dirección j.

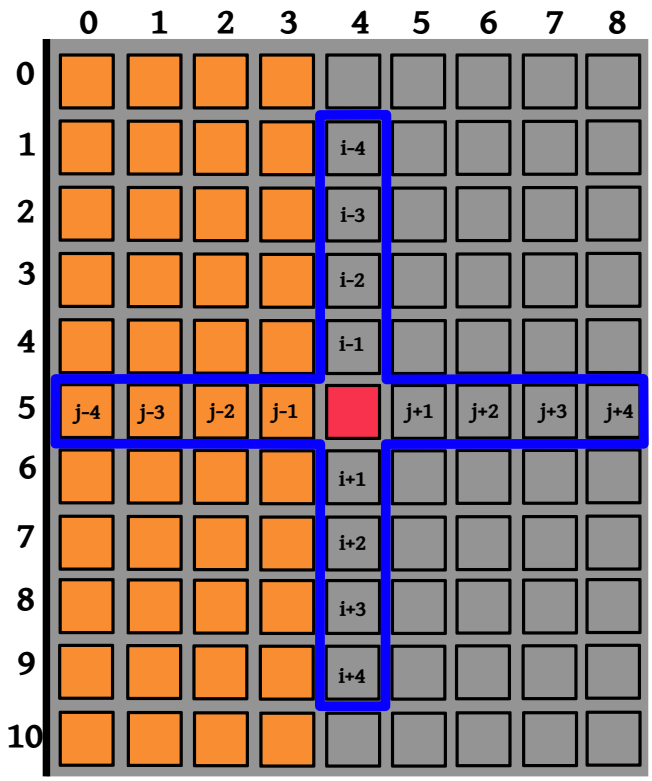


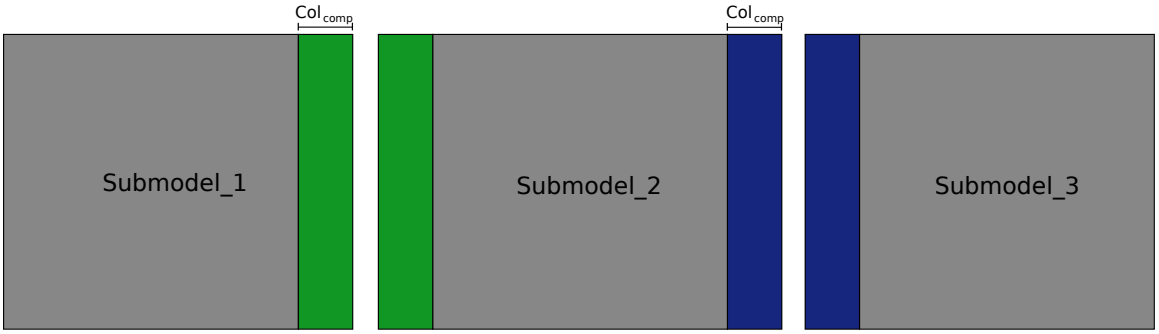
Figura 12: Planteamiento de la malla de diferencias finitas de octavo orden en la frontera izquierda del modelo.

Para realizar la implementación Se debe describir la primera condición de la estrategia, la cual es la división del modelo de velocidades. La división se procura realizar en partes iguales en la dimensión que tenga el mayor tamaño e intentando que los submodelos sean lo más cuadrado posibles (cúbico en el caso 3D) para así obtener la mejor distribución de memoria. Adicionalmente, teniendo en cuenta el problema de las fronteras, estos submodelos compartirán valores entre sí y de manera adyacente para que los cortes de los modelos no exista este problema. El número de columnas compartidas está regida por:

$$Col_{comp} = \frac{OrdenFD}{2} \ , \tag{2.1}$$

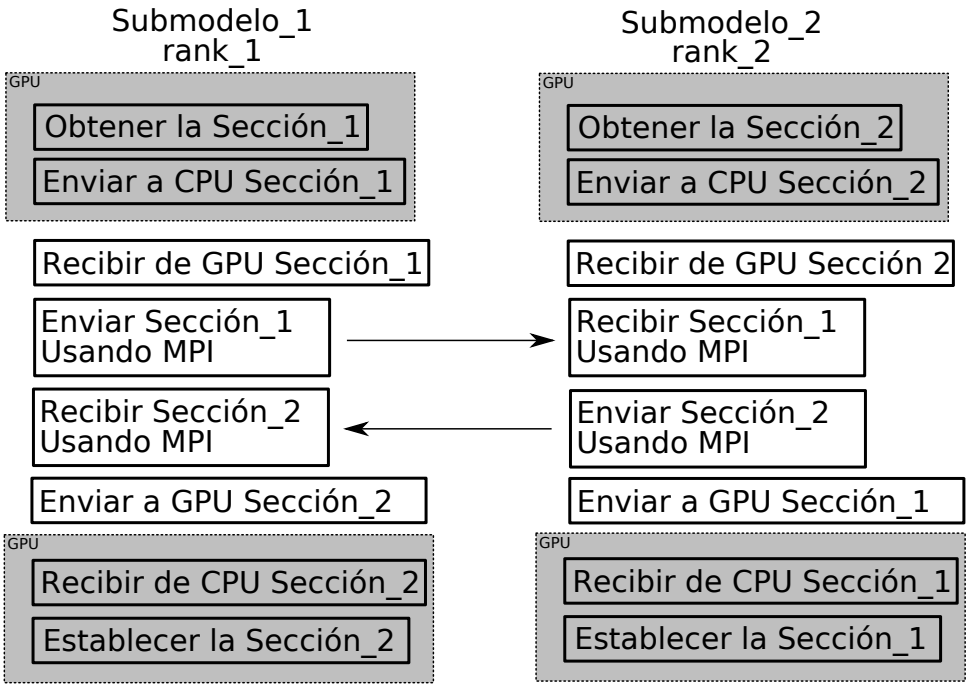
donde $OrdenFD$ es el orden de la aproximación de diferencias finitas espaciales. Esta ecuación viene dada por la restricción de las fronteras que se explicó anteriormente en la figura 12. En la figura 13 se observa las franjas de los submodelos que se compartirán entre sí.

Figura 13: Secciones compartidas entre los submodelos.



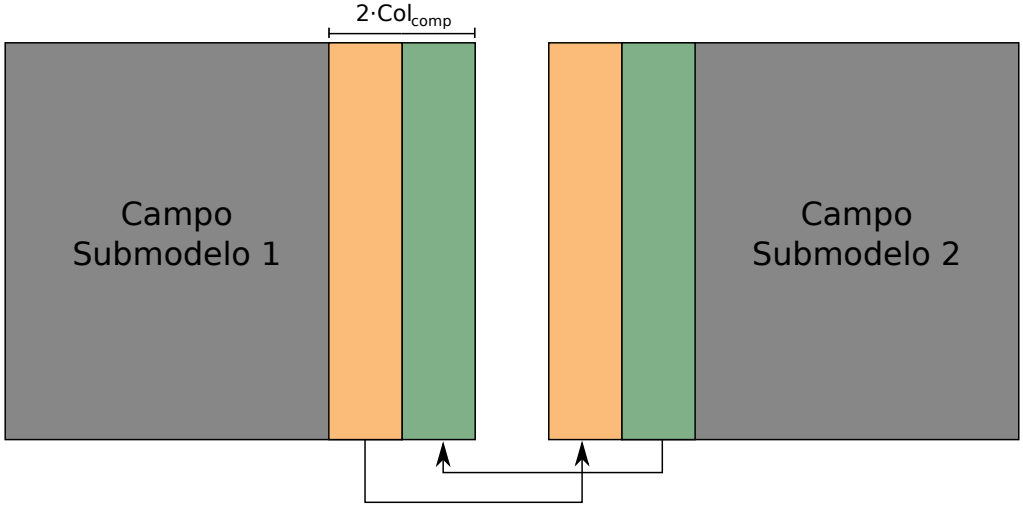
Una vez obtenidos los submodelos, se aplica la segunda condición, la cual es realizar la propagación de onda en cada uno de ellos, efectuando en cada iteración de tiempo una operación de *swapping*. Esta operación consiste en intercambiar secciones del campo calculado entre los modelos adyacentes para que la continuidad del fenómeno físico se mantenga entre todos los submodelos. El esquema de funcionamiento de la operación de *swapping* se puede ver en la figura 14, donde se observa que el intercambio entre submodelos se realiza de forma lineal y siendo esta la sección más lenta del algoritmo.

Figura 14: Operación de *swapping* entre dos modelos adyacente.



En la figura 15 se observa que secciones del campo se intercambian y en que posiciones se establecen para que funcionen como condiciones iniciales.

Figura 15: Operación de *swapping* entre dos modelos adyacente. Los colores indican las secciones que serán intercambiadas y las flechas la dirección del intercambio.



Después de implementada la estrategia sobre el módulo de propagación de onda usando descomposición de dominio, se realiza el proceso de migración de manera tradicional, usando en cada GPU su respectiva porción de modelo. Una vez terminado y generado sus porciones de imágenes, se unen los dos resultados teniendo en cuenta la primera condición para así obtener la imagen final completa.

3. MÉTRICAS Y RESULTADOS

En este capítulo se describen los tipos de pruebas que se aplicaron a las implementaciones realizadas, sus respectivos resultados, comparaciones entre sí y su discusión. El primer factor que se tuvo en cuenta fue la comprobación del correcto funcionamiento del algoritmo implementado usando las estrategias propuestas. Luego se midieron los tiempos de cómputo para obtener conclusiones de eficiencia de las estrategias. La tercera métrica que se tuvo en cuenta fue la ocupación de memoria en los dispositivos GPU ya que esta es un factor crítico en la implementación. Por último, se realizó el análisis de la complejidad computacional de las implementaciones para tener un referente adicional a la medición de tiempos, y así poder estimar una estimación del comportamiento de la implementación al aumentar el tamaño de valores a procesar.

Las pruebas se realizaron sobre un mismo modelo de velocidades (figura 16) y se observaron secciones planas del modelo para comprobar el correcto funcionamiento del algoritmo (Figura 17). Los parámetros que se utilizaron para las pruebas se listan a continuación:

- $ns = 60$. Número de disparos realizados.
- $nx = 169$ [puntos]. Dimensión en x (*Inline*).
- $ny = 169$ [puntos] . Dimensión en y (*Crossline*).
- $nz = 80$ [puntos]. Dimensión en z (*Depth*).
- $dh = 20$ [m]. Paso espacial.
- $tend = 2{,}5$ [s]. Tiempo de grabación de los geófonos.
- $dt = 0{,}002$ [s]. Paso temporal.
- $fq = 10$ [Hz]. Frecuencia de la ondícula del disparo.
- $ng = 1 : 4$. Número de GPUs usadas.

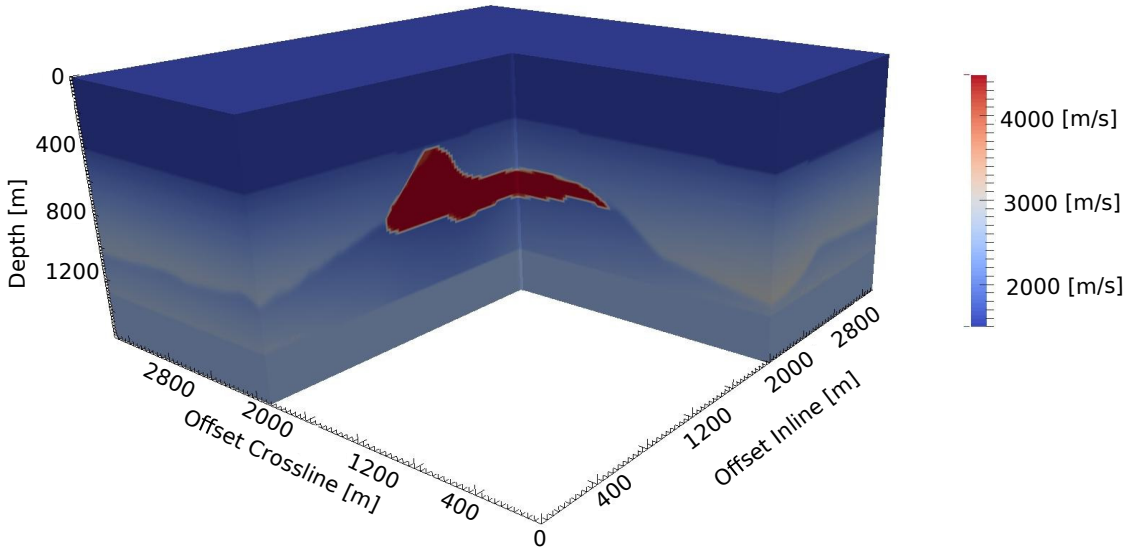


Figura 16: Modelo *SEG/EAGE Salt Model* utilizado para las pruebas.

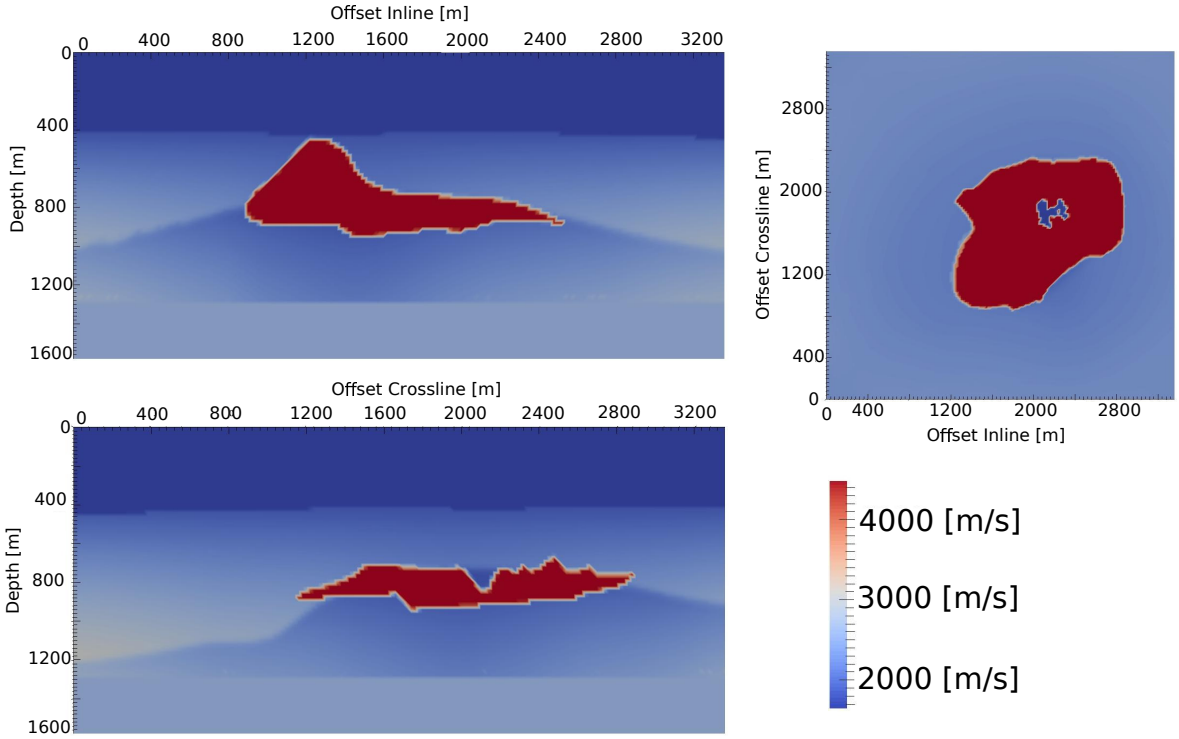


Figura 17: Secciones planas del Modelo *SEG/EAGE Salt Model* utilizado para las pruebas.

Este modelo de velocidades 3D fue creado en conjunto por dos sociedades de geociencias, la SEG (*Society of Exploration Geophysicists*) y la EAGE (*European Association of Geoscientists and Engineers*) [17]. El modelo posee características especiales que deben ser vistas cuando se realiza una migración sobre él. La principal característica es el domo de sal que se encuentra en medio del modelo y tiene una forma como se ve en la figura 18 el cual posee una alta velocidad comparada a sus capas adyacentes.

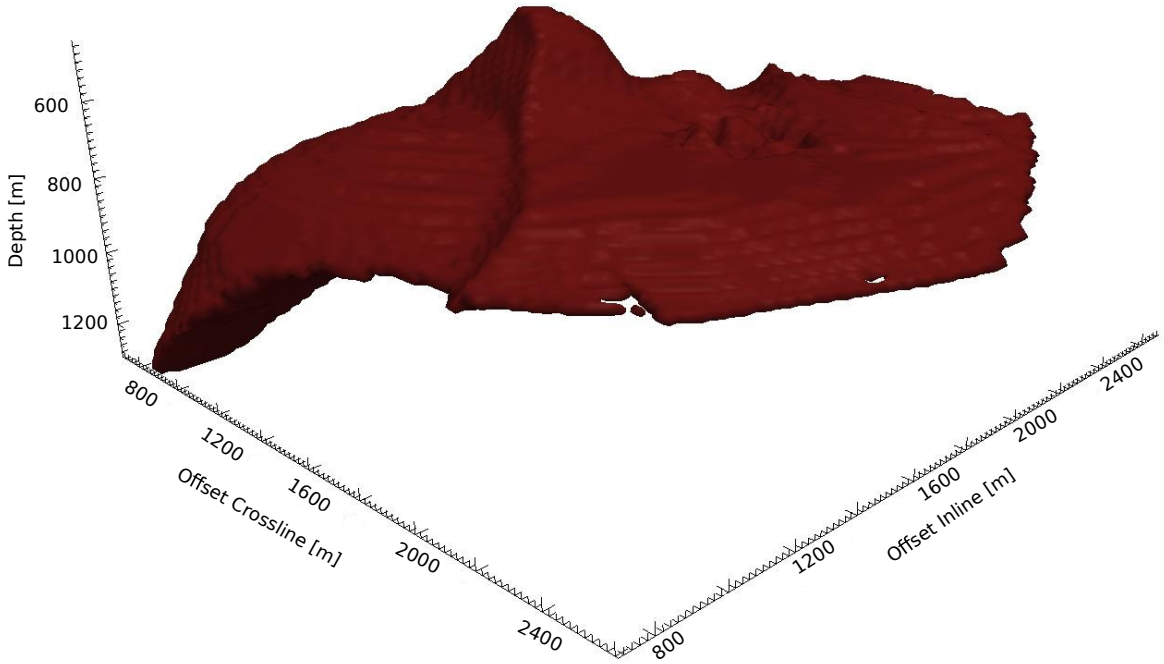


Figura 18: Domo de sal extraído del modelo de velocidades *SEG/EAGE Salt Model*.

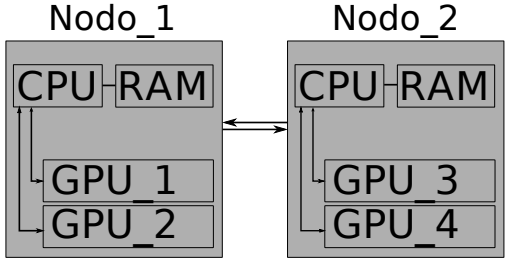
3.1. EQUIPO DE CÓMPUTO USADO

Las mediciones de los parámetros de tiempo y memoria están sujetos al hardware del equipo de cómputo, por esta razón en las pruebas que se realizaron, se procuró tener una medición lo mas similar entre ellas sin modificar el Hardware del *cluster*. La figura 19 muestra la distribución del *cluster* utilizado. Las especificaciones de Hardware de cada nodo de procesamiento se detalla en la tabla 2.

Tabla 2: Detalle de los nodos utilizados.

	Caracteristica	Detalle
Nodo_1	CPU	Intel Xeon E5-2620 2.40 GHz
	CPU_{RAM}	≈256 [GB]
	$GPUx2$	Tesla K40m
	GPU_{RAM}	≈12 [GB]
Nodo_2	CPU	Intel Xeon E5-2620 2.40 GHz
	CPU_{RAM}	≈256 [GB]
	$GPUx2$	Tesla K40m
	GPU_{RAM}	≈12 [GB]

Figura 19: Esquema del cluster utilizado para la pruebas.



3.2. RESULTADOS DE MIGRACIÓN USANDO REVERSE TIME MIGRATION 3D

Para comprobar si las imágenes que se están generando usando las estrategias son correctas comparadas con la implementación de referencia (implementación en una GPU), se realizó el cálculo de la norma l_2 sobre la diferencia entre la imagen obtenida de referencia con la imagen obtenida por cada uno de las estrategias definida así:

$$\|Img\|_{l_2} = \sum_{i=0}^{n} \sqrt{(ImgRef_i - ImgStr_i)^2}, \tag{3.1}$$

donde n es el número de elementos que tiene la imagen. El cálculo de las normas se puede ver en la tabla 3 y se concluyó que las tres imágenes son muy similares ya que su norma l_2 se acerca a cero, esto es de esperar ya que el algoritmo es el mismo y la única variante es la forma en como se implementa. La Figura 20 muestra las secciones planas de las imágenes migradas usando las estrategias propuestas.

Tabla 3: Norma l_2 entre las imágenes generadas usando las estrategias propuestas y la implementación de referencia.

	$\|Img\|_{l_2}$
Estrategia 1	0.0030
Estrategia 2	0.0

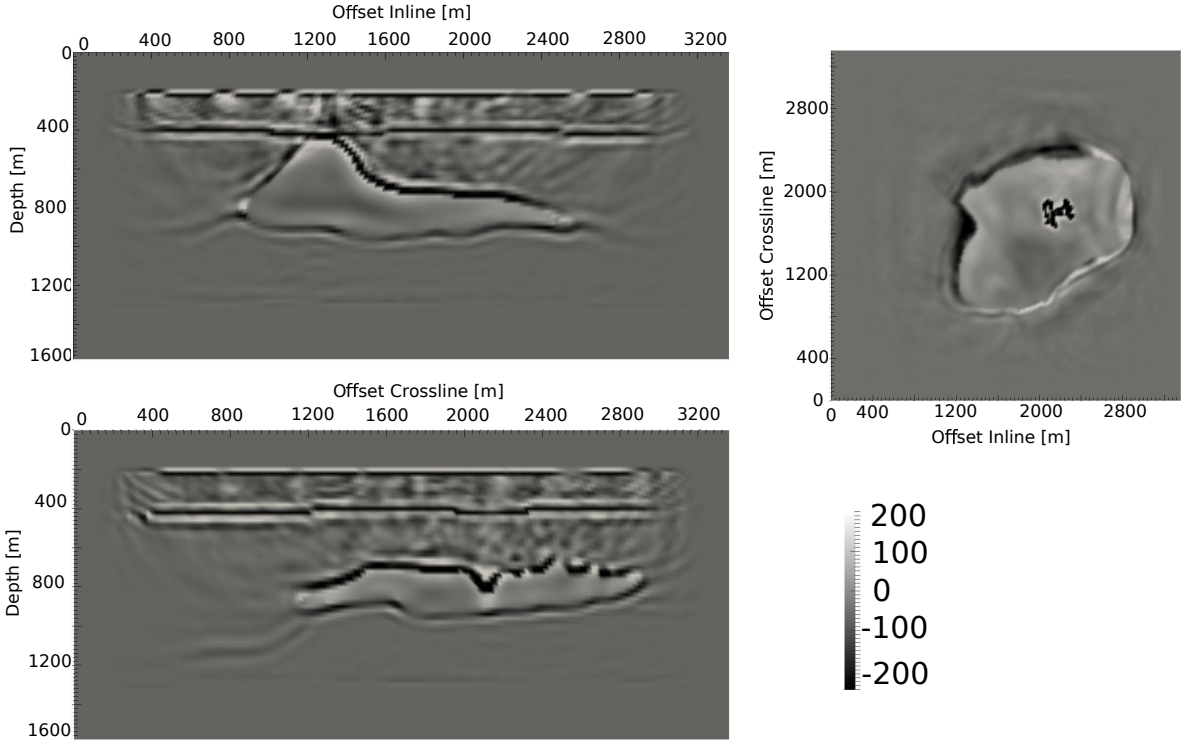


Figura 20: Secciones planas de la imagen final migrada usando RTM.

3.3. MEDICIONES DE TIEMPO

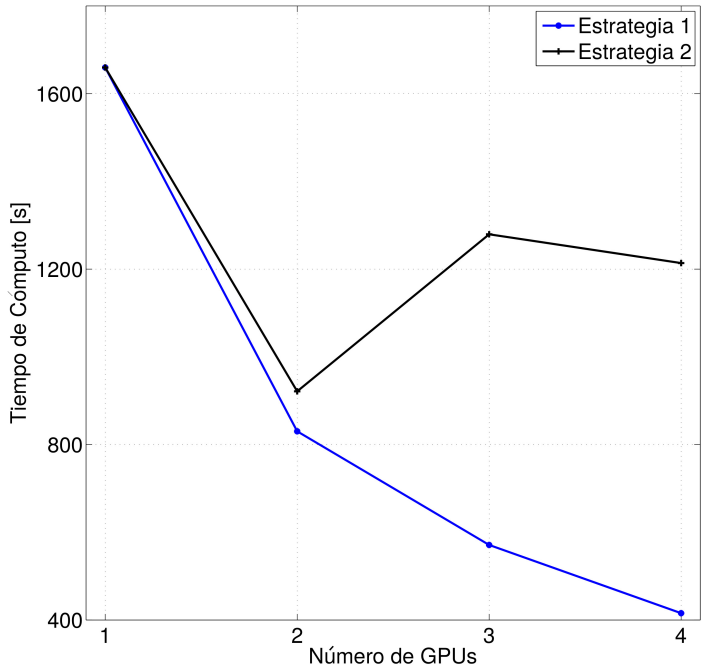
Los tiempos de cómputo se midieron en las tres implementaciones: RTM en una GPU (referencia) y a las dos estrategias propuestas. En la tabla 4 se observan los tiempos medidos en segundos de las implementaciones desde 1 hasta 4 GPUs contrastados con la implementación de referencia.

Tabla 4: Tiempos medidos usando las estrategias 1 y 2.

	1 GPU (Referencia)	2 GPU	3 GPU	4 GPU
Estrategia 1	1659.64 [s]	830.24 [s]	571.30 [s]	415.80 [s]
Estrategia 2	1659.64 [s]	921.37 [s]	1279.42 [s]	1213.78 [s]

Para resumir la tabla 4 se realizó la figura 21 donde se puede observar que ambas estrategias tiene el mismo valor de partida que es el tiempo de referencia. Para la estrategia 1 se puede ver una evidente reducción en los tiempos de cómputo y que estos decrecen. En la estrategia 2 se puede detallar que hay una reducción en los tiempos de cómputo cuando se usa 2 GPUs, mientras que para 3 y 4 GPUs estos tiempos aumentan pero sin rebasar el tiempo referencia.

Figura 21: Tiempos de cómputo obtenidos por las estrategias 1 y 2.



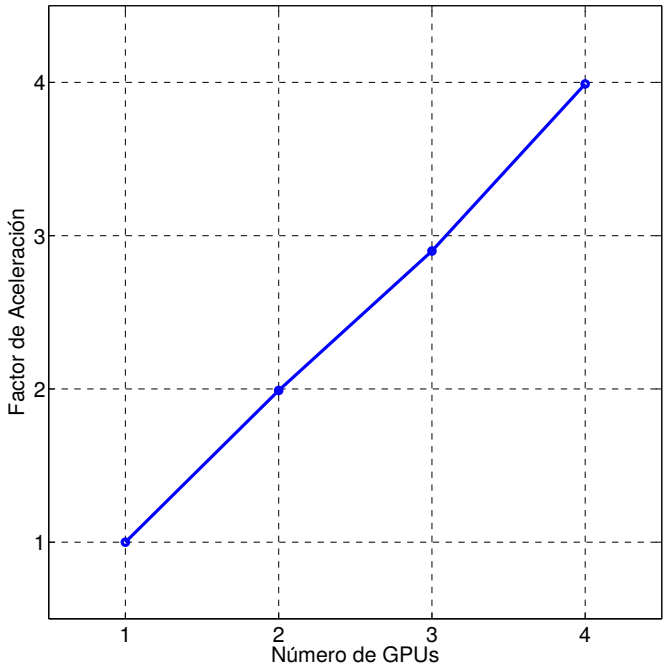
3.4. ANÁLISIS DE TIEMPOS DE LA ESTRATEGIA 1

Como se ve en la figura 21 existe una reducción de tiempos de cómputo al utilizar la estrategia 1, además se puede ver que su línea de tendencia es decreciente, esto quiere decir que existirá un valor máximo de GPUs donde la implementación dejara de reducir tiempos. Se realizó el cálculo del factor de aceleración que viene dado por

$$Factor\ de\ aceleración = \frac{t_{ref}}{t_{est}}\ , \tag{3.2}$$

donde t_{ref} y t_{est} son los tiempos de referencia y los de la estrategia implementada respectivamente. Se calculó éste factor para todos los casos medidos obteniendo la Figura 22, donde se observa que la línea de tendencia del factor de aceleración es lineal llegando a obtener un factor de aceleración de 3,99 en el caso de usar 4 GPUs acercándose a su valor ideal el cual es 4, es decir, el número de GPUs que se usen.

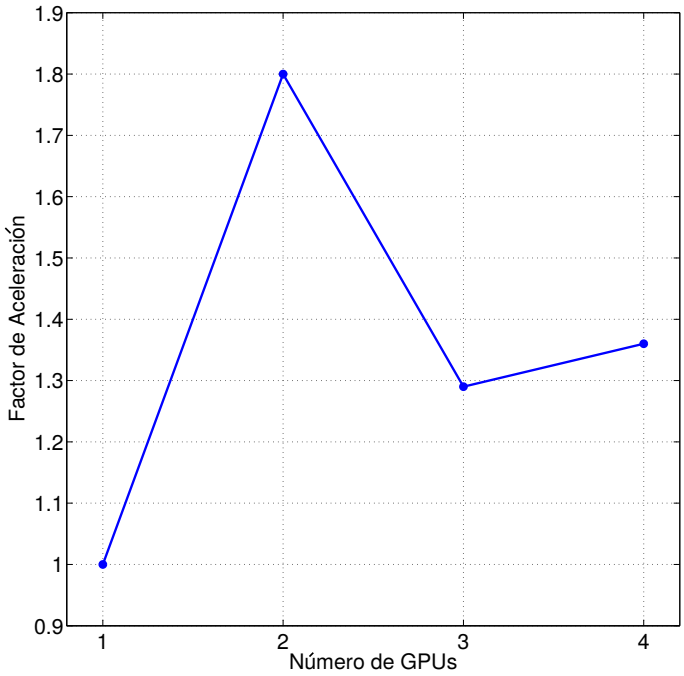
Figura 22: Factor de aceleración obtenido usando la estrategia 1.



3.5. ANÁLISIS DE TIEMPOS DE LA ESTRATEGIA 2

Se realizó el cálculo del factor de aceleración usando la ecuación 3.2 y se obtuvo la figura 23, donde se puede observar que esta no tiene una línea de tendencia clara ya que el número de GPUs es limitado. Además, también se puede ver que usando 2 GPUs el factor de aceleración es máximo para esta implementación y que después comienza a disminuir al aumentar el número de GPUs, esto es debido que a un mayor número de GPUs conlleva un mayor número de trasmisiones entre los submodelos.

Figura 23: Factor de aceleración obtenido usando la estrategia 2.



3.6. OCUPACIÓN DE MEMORIA

La segunda medida que se tuvo en cuenta para comparar las estrategias propuestas fue la ocupación de memoria por GPU, para esto se realizaron mediciones experimentales y cálculos teóricos para cada una de las 3 implementaciones. Para las implementaciones de referencia y usando la estrategia 1, se tiene la misma ecuación de memoria ya que estas dos comparten la característica de procesar un disparo a la vez y su ecuación de memoria teórica por GPU es:

$$\begin{aligned} Mem_1 = &[13 \cdot (nx \cdot ny \cdot nz) + 2 \cdot (nx \cdot ny \cdot nt) + \\ &2 \cdot nt + (nx \cdot ny \cdot nz \cdot nt)] \cdot \frac{4}{1024^2} + 77 \ [MB] \ , \end{aligned} \tag{3.3}$$

donde el número 77 es un término constante que la GPU reserva por defecto para su funcionamiento interno. La ecuación de memoria para la estrategia 2 es:

$$\begin{aligned} Mem_2 = &[12 \cdot (nx \cdot ny_{mod} \cdot nz) + 2 \cdot (nx \cdot ny_{mod} \cdot nt) + \\ &3 \cdot nt + (nx \cdot ny_{mod} \cdot nz \cdot nt)] \cdot \frac{4}{1024^2} + 77 \ [MiB] \ , \end{aligned} \tag{3.4}$$

donde ny_{mod} es la dimensión que se esta descomponiendo y tendrá un tamaño dependiendo de la posición donde esté el submodelo. El valor del tamaño de ny_{mod} para cada submodelo viene regida por las siguientes condiciones:

$$ny_{mod} = \begin{cases} ceil(\frac{ns}{ng}) + 4 & \text{Si el submodelo está en los extremos.} \\ ceil(\frac{ns}{ng}) + 8 & \text{Si el submodelo está en medio de los extremos.} \end{cases} \tag{3.5}$$

3.7. ANÁLISIS DE OCUPACIÓN DE MEMORIA DE LA ESTRATEGIA 1

Se realizaron las mediciones y los cálculos de los valores de ocupación de memoria comparando la implementación de referencia con la implementación de la estrategia 1 usando el máximo de GPUs con los parámetros descritos al inicio de este capítulo, los resultados se pueden ver en la tabla 5.

Tabla 5: Medición de ocupación de memoria para la implementación de referencia y la estrategia 1.

	Id de GPU	Memoria Teórico	Memoria Experimental
Referencia	1	11.33 [GB]	11.35 [GB]
Estrategia 1	1	11.33 [GB]	11.35 [GB]
	2	11.33 [GB]	11.35 [GB]
	3	11.33 [GB]	11.35 [GB]
	4	11.33 [GB]	11.35 [GB]

De la tabla 5 se puede observar que la ocupación de memoria entre las implementaciones es la misma para todas las GPUs, ya que ambas implementaciones fueron programadas para procesar un disparo (*shot*) a la vez, concluyendo que una restricción para la estrategia 1 es que se debe asegurar que al menos un disparo pueda ser alojado en la GPU, de lo contrario esta estrategia no podrá ser implementada.

Tabla 6: Medición de ocupación de memoria para la implementación de referencia y la estrategia 2.

	# de GPUs	Memoria Teórico	Memoria Experimental
Referencia	1	11.33 [GB]	11.35 [GB]
Estrategia 2	2	5.96 [GB]	5.97 [GB]
	3	4.18 [GB]	4.18 [GB]
	4	3.28 [GB]	3.29 [GB]

3.8. ANÁLISIS DE OCUPACIÓN DE MEMORIA DE LA ESTRATEGIA 2

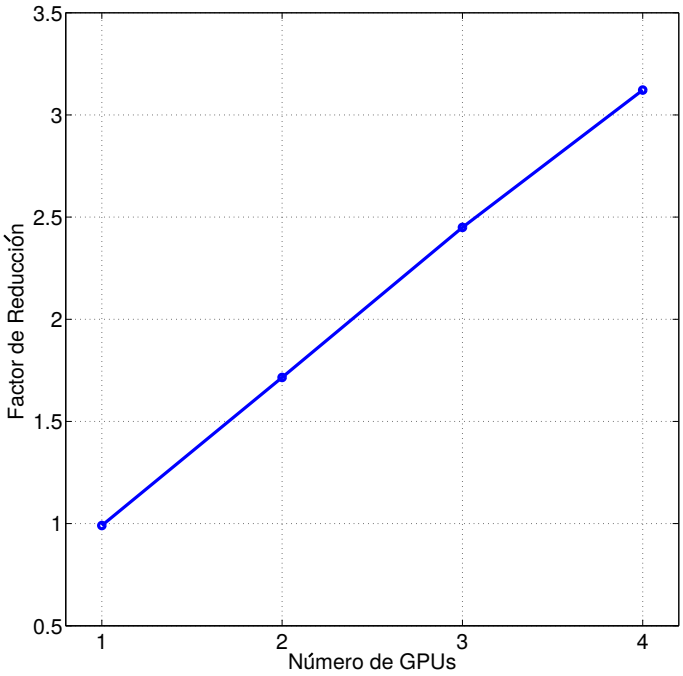
Se realizaron las medidas de la ocupación de memoria para la estrategia 2, tomando el valor promedio de memoria que usa cada GPU, utilizando desde 2 a 4 GPUs y se comparó nuevamente con la implementación de referencia. Los resultados de pueden ver en la tabla 6.

La comparación más notable entre la implementación de referencia y la estrategia 2 es la reducción de memoria por GPU, ya que al usar una mayor cantidad de GPUs, la memoria se reduce haciendo que esta estrategia mitigue el impacto del problema de memoria del algoritmo RTM. Se calculó el factor de reducción de memoria dada por

$$Factor\ de\ reduccion = \frac{mem_{ref}}{mem_{est}}\ , \tag{3.6}$$

donde mem_{ref} es el el valor de memoria usada en la implementación de referencia y mem_{est} es el valor obtenido de memoria usando la estrategia 2. Se calcularon estos factores y se obtuvo la figura 24, donde se puede observar que su tendencia es lineal y que su máximo factor de reducción de memoria para esta implementación fue de 3,12.

Figura 24: Factor de reducción de memoria usando la estrategia 2.



3.9. COMPLEJIDAD COMPUTACIONAL

La complejidad computacional estudia los recursos requeridos durante el cálculo de la solución a un problema, la complejidad temporal se encarga de observar y concluir el nivel de dificultad que tiene un determinado algoritmo comparado con otro [18]. Se calculó la complejidad computacional para las tres implementaciones empezando por la de referencia. La sección crítica de algoritmo [5] para calcular la complejidad computacional de la implementación de referencia se ve en el algoritmo 1. Tomando como referencia este algoritmo se puede determinar que la complejidad computacional en notación *Big-O* es:

$$O(ns \cdot nt \cdot ny). \tag{3.7}$$

Para el cálculo de la complejidad computacional de la implementación de la estrategia 1, se tomó como referencia el algoritmo 1, ya que estas dos implementaciones son similares exceptuando que en el primer ciclo *For* su final no será ns sino ns/ng donde ng es el número de GPUs que se utilicen en la implementación, esta variante no indica un

[5] Segmento del algoritmo principal el cual realiza más cómputo intensivo.

Algoritmo 1 Algoritmo crítico de la implementación de referencia.

```
1: for (shots = 0, ··· , ns) do                          ▷ ns, número de disparos
2:     for steptime = 0, ··· , nt do                     ▷ it, número de pasos de tiempo
3:         function Propagador3D()
4:             for dimY = 0, ··· , ny do                 ▷ ny, tamaño de la dimensión en Y
5:             end for
6:         end function
7:     end for
8: end for
```

cambio en la complejidad computacional ya que el número de GPUs será una constante y por lo tanto se elimina, llegando a la conclusión que la complejidad computacional de la estrategia 1 es:

$$O\left(ns \cdot nt \cdot ny\right). \tag{3.8}$$

En el caso de la estrategia 2, existen cambios en el algoritmo crítico y este se observa a continuación:

Algoritmo 2 Algoritmo crítico de la implementación de de la estrategia 2.

```
1: for (shots = 0, ··· , ns) do                          ▷ ns, número de disparos
2:     for steptime = 0, ··· , nt do                     ▷ it, número de pasos de tiempo
3:         function Propagador3D()
4:             for dimY = 0, ··· , ny do                 ▷ ny, tamaño de la dimensión en Y
5:             end for
6:         end function
7:
8:         function Swapping()
9:             for GPUs = 0, ··· , ng do
10:                for block = 0, ··· , (4 · nx · nz) do
11:                end for
12:            end for
13:        end function
14:    end for
15: end for
```

A partir del algoritmo 2 se puede calcular la complejidad computacional de la estrategia 2. En este algoritmo existen dos llamados a funciones (*Propagador3D y Swapping*) donde este último es el mas crítico ya que realiza envíos de datos y este puede ser visto

como un ciclo de $nx \cdot nz \cdot ng$ iteraciones. La complejidad computacional de la estrategia 2 es la siguiente:

$$O(\underbrace{ns \cdot nt \cdot ng \cdot nx \cdot nz}_{\text{Swapping}} + \underbrace{ns \cdot nt \cdot ny}_{\text{Propagador3D}}). \tag{3.9}$$

Con estas tres complejidades computacionales calculadas se puede concluir que entre la estrategia 1 y la implementación de referencia su complejidad se mantiene constante, esto se esperaba ya que el algoritmo se comporta de la misma manera exceptuando que el cómputo se reparte entre más GPUS. Otra abstracción que se puede realizar es que la estrategia 2 es mas compleja, puesto que esta requiere hacer operaciones de transmisión de datos entre las GPUs.

4. CONCLUSIONES

Se realizó la implementación del algoritmo *Reverse Time Migration 3D* sobre una GPU, midiendo tiempos de procesado, ocupación de memoria y calculando su complejidad ocupacional del algoritmo, además se obtuvo una imagen final migrada satisfactoria, visualmente y de magnitudes correcta de un modelo 3D.

Se hizo la búsqueda de estándares de comunicación que tuvieran la característica de poder controlar e intercambiar información entre dispositivos GPU, donde MPI (*Message Passing Interface*) fue el más apto para las necesidades de la implementación en un *cluster* GPU, porque su protocolo de comunicación es el más robusto ya que posee una topología virtual, permitiendo generar patrones entre los procesos que se generen, haciendo que la eficiencia del sistema de comunicación aumente.

Se plantearon dos estrategias de implementación donde se usó un *cluster* con múltiples GPUs. La primera de ellas, migración sísmica por división de disparos, logro obtener una reducción de tiempos de cómputo alcanzando un factor de aceleración máximo de 3.99 muy cercano al valor de teórico. Además este factor de aceleración se comporto de manera lineal indicando que el tiempo de cómputo es mucho mayor que el tiempo de transmisión final que se requiere para generar la imagen final. También para esta estrategia se logró establecer una condición de funcionamiento, la cual es que se debe asegurar que un disparo pueda ser alojado en una GPU, de lo contrario esta estrategia no podrá ser implementada.

La segunda estrategia que se implementó se llamó migración sísmica por descomposición de dominio y se obtuvieron mejoras en los tiempos de cómputo debido que la división del modelo en porciones mas pequeñas provocaba que la carga computacional por GPU se redujera. Cuando se aumentó el número de GPUs los tiempos de cómputo aumentaron ya que se realiza un mayor número de intercambios de información (operación *swapping*) y este se realiza de manera semi-serial. La conclusión mas notoria de esta estrategia es su reducción en la ocupación de memoria por GPU, dividiendo el modelo de velocidades entre las GPUs implementadas y logrando así que los campos calculados se reduzcan y obteniendo un factor de reducción de 3.12.

Teniendo en cuenta las conclusiones obtenidas con este trabajo de investigación se puede plantear un trabajo futuro con el objetivo ayudar en el mejoramiento de estas estrategias. Observando los dos aportes que hacen las estrategias implementadas, que son la reducción de tiempos de cómputo y la reducción de la ocupación de memoria, se podría realizar una implementación donde se unan estas dos obteniendo ambos beneficios. Esta implementación necesitaría un mayor numero de GPUs porque la división de disparos ya no se realizaría entre GPUs sino entre clusters o grupo de GPUs.

REFERENCIAS

[1] Lore Gary L, Marin David A, Batchelder Eric C, Courtwright William C, Desselles Jr Richard P, and Klazynski Ralph J, "2000 assessment of conventionally recoverable hydrocarbon resources of the gulf of mexico and atlantic outer continental shelf as of january 1, 1999," *US Department of the Interior, Minerals Management Service, Gulf of Mexico OCS Region, Office of Resource Evaluation, New Orleans, LA*, 2001.

[2] Araya Polo Mauricio and Rubio Felix, "High-performance seismic acoustic imaging by reverse-time migration on the cell/b.e. architecture," *Barcelona*, 2003.

[3] Baysal Edip, Kosloff Dan D, and Sherwood John WC, "Reverse time migration," *Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983.

[4] Foltinek Darren, Eaton Daniel, Mahovsky Jeff, Moghaddam Peyman, McGarry Ray, *et al.*, "Industrial-scale reverse time migration on gpu hardware," en *2009 SEG Annual Meeting*, Society of Exploration Geophysicists, 2009.

[5] Panetta Jairo, Teixeira Thiago, de Souza Filho Paulo RP, da Cunha Finho Carlos A, Sotelo David, da Motta Fernando M Roxo, Pinheiro Silvio Sinedino, Junior Ivan Pedrosa, Rosa Andre L Romanelli, Monnerat Luiz R, *et al.*, "Accelerating kirchhoff migration by cpu and gpu cooperation," en *Computer Architecture and High Performance Computing, 2009. SBAC-PAD'09. 21st International Symposium on*, pp. 26–32, IEEE, 2009.

[6] Amado Jhonatan, Salamanca William, Vivas Flor Alba, and Ramirez Ana, "A gpu implementation of the reverse time migration algorithm," en *14th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 3-6 August 2015*, pp. 1016–1021, Brazilian Geophysical Society, 2015.

[7] Sullivan Dennis M, *Electromagnetic simulation using the FDTD method*. John Wiley & Sons, 2013.

[8] Pasalic Damir, McGarry Ray, *et al.*, "Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations," 2010.

[9] Gropp William, Hoefler Torsten, Thakur Rajeev, and Lusk Ewing, *Using advanced MPI: Modern features of the message-passing interface*. MIT Press, 2014.

[10] Nichols Bradford, Buttlar Dick, and Farrell Jacqueline, *Pthreads programming: A POSIX standard for better multiprocessing*. .°'Reilly Media, Inc.", 1996.

[11] Lewis Bil and Berg Daniel J, "Pthreads primer," *Sun Microsystems Inc*, 1996.

[12] Geist Al, *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994.

[13] "Introduction cuda aware mpi." `https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/`, 2013. [Online; accedido 20-06-2016].

[14] Yilmaz Oz, *Seismic Data Analysis: Processing, Inversion, and Interpretation of Seismic Data*. Society of Exploration Geophysicists., 2001.

[15] Moussa Nader, "Seismic imaging using gpgpu accelerated reverse time migration," *CS 315A Parallel Computer Architecture and Programming*, 2009.

[16] Villarreal Alberto and Scales John A, "3d finite difference modeling via domain decomposition," en *SEG Technical Program Expanded Abstracts 1996*, pp. 1231–1234, Society of Exploration Geophysicists, 1996.

[17] Bulant P, "Constructing the seg/eage 3-d salt model for ray tracing using sobolev scalar products," *Studia Geophysica et Geodaetica*, vol. 48, no. 4, pp. 689–707, 2004.

[18] Cortez Augusto, "Teoría de la complejidad computacional y teoría de la computabilidad," *Universidad Nacional Mayor de San Marcos. Lima, Perú*, 2004.

[19] "CUDA C Programming Guide." `http://docs.nvidia.com/cuda/`, 2015. [Online; accedido 20-06-2016].

BIBLIOGRAFÍA

Araya Polo Mauricio and Rubio Felix, "High-performance seismic acoustic imaging by reverse-time migration on the cell/b.e. architecture," *Barcelona*, 2003

Amado Jhonatan, Salamanca William, Vivas Flor Alba, and Ramirez Ana, "A gpu implementation of the reverse time migration algorithm," en *14th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 3-6 August 2015*, pp. 1016–1021, Brazilian Geophysical Society, 2015

Baysal Edip, Kosloff Dan D, and Sherwood John WC, "Reverse time migration," *Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983

Bulant P, "Constructing the seg/eage 3-d salt model for ray tracing using sobolev scalar products," *Studia Geophysica et Geodaetica*, vol. 48, no. 4, pp. 689–707, 2004

"CUDA C Programming Guide." `http://docs.nvidia.com/cuda/`, 2015. [Online; accedido 20-06-2016]

Cortez Augusto, "Teoría de la complejidad computacional y teoría de la computabilidad," *Universidad Nacional Mayor de San Marcos. Lima, Perú*, 2004

Foltinek Darren, Eaton Daniel, Mahovsky Jeff, Moghaddam Peyman, McGarry Ray, *et al.*, "Industrial-scale reverse time migration on gpu hardware," en *2009 SEG Annual Meeting*, Society of Exploration Geophysicists, 2009

Geist Al, *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994

Gropp William, Hoefler Torsten, Thakur Rajeev, and Lusk Ewing, *Using advanced MPI: Modern features of the message-passing interface*. MIT Press, 2014

"Introduction cuda aware mpi." `https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/`, 2013. [Online; accedido 20-06-2016]

Lewis Bil and Berg Daniel J, "Pthreads primer," *Sun Microsystems Inc*, 1996

Lore Gary L, Marin David A, Batchelder Eric C, Courtwright William C, Desselles Jr Richard P, and Klazynski Ralph J, "2000 assessment of conventionally recoverable hydrocarbon resources of the gulf of mexico and atlantic outer continental shelf as of january 1, 1999," *US Department of the Interior, Minerals Management Service, Gulf of Mexico OCS Region, Office of Resource Evaluation, New Orleans, LA*, 2001

Moussa Nader, "Seismic imaging using gpgpu accelerated reverse time migration," *CS 315A Parallel Computer Architecture and Programming*, 2009

Nichols Bradford, Buttlar Dick, and Farrell Jacqueline, *Pthreads programming: A POSIX standard for better multiprocessing.* " O'Reilly Media, Inc.", 1996

Panetta Jairo, Teixeira Thiago, de Souza Filho Paulo RP, da Cunha Finho Carlos A, Sotelo David, da Motta Fernando M Roxo, Pinheiro Silvio Sinedino, Junior Ivan Pedrosa, Rosa Andre L Romanelli, Monnerat Luiz R, *et al.*, "Accelerating kirchhoff migration by cpu and gpu cooperation," en *Computer Architecture and High Performance Computing, 2009. SBAC-PAD'09. 21st International Symposium on*, pp. 26–32, IEEE, 2009

Pasalic Damir, McGarry Ray, *et al.*, "Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations," 2010

Sullivan Dennis M, *Electromagnetic simulation using the FDTD method.* John Wiley & Sons, 2013

Villarreal Alberto and Scales John A, "3d finite difference modeling via domain decomposition," en *SEG Technical Program Expanded Abstracts 1996*, pp. 1231–1234, Society of Exploration Geophysicists, 1996

Yilmaz Oz, *Seismic Data Analysis: Processing, Inversion, and Interpretation of Seismic Data.* Society of Exploration Geophysicists., 2001