ESTRATEGIA COMPUTACIONAL DE IMPLEMENTACIÓN DE LA INVERSIÓN DE ONDA COMPLETA 3D PARA DATOS REALES USANDO UN CLUSTER DE GPUS

REYNALDO FABIAN NORIEGA ZAMBRANO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2018

ESTRATEGIA COMPUTACIONAL DE IMPLEMENTACIÓN DE LA INVERSIÓN DE ONDA COMPLETA 3D PARA DATOS REALES USANDO UN CLUSTER DE GPUS

REYNALDO FABIAN NORIEGA ZAMBRANO

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

Director
Dr. Ing. Sergio Alberto Abreo Carrillo
Co Director
PhD. Ana Beatriz Ramírez Silva

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2018

DEDICATORIA

A mi madre Rosmery Zambrano, por ser el pilar fundamental de mi vida por su apoyo incondicional y esmero en brindarme lo mejor.

A mi hermana y amiga Maria Fernanda, por brindarme la confianza y motivación necesarias para seguir adelante.

A Estefany Pájaro, por su cariño, por estar a mi lado en esta etapa de mi vida y su paciencia en los momentos más difíciles.

AGRADECIMIENTOS

Este trabajo fue desarrollado con el apoyo de ECOPETROL durante el proyecto de investigación No. 0266-2013.

Agradezco a los docentes de la escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones de la Universidad Industrial de Santander que fueron participantes en mi formación profesional.

Un agradecimiento general al grupo de investigación CPS y todos sus miembros. A mis directores de proyecto, los docentes Sergio Abreo y Ana Ramírez, por su apoyo en el desarrollo de este trabajo.

Un agradecimiento a mis amigos Fabian Sanchez y David Abreo por ser unas personas admirables y de las cuales aprendí mucho en este proceso.

Por último, un agradecimiento a Fabian Noriega. Claro que sí, al final, fui yo quién se esforzó en alcanzar esta meta (mi libro, mis agradecimientos y me los follo cuando quiera :D).

Reynaldo Fabian Noriega Zambrano

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

TÍTULO: ESTRATEGIA COMPUTACIONAL DE IMPLEMENTACIÓN DE LA INVERSIÓN DE ONDA COMPLETA 3D PARA DATOS REALES USANDO UN CLUSTER DE GPUS [*]

AUTOR: REYNALDO FABIAN NORIEGA ZAMBRANO [**]

PALABRAS CLAVE: INVERSIÓN DE ONDA COMPLETA 3D, FDTD

DESCRIPCIÓN:

La inversión de onda completa (FWI, por sus siglas en inglés) es un proceso iterativo que estima características del subsuelo (velocidad o densidad) mediante la minimización de la diferencia entre los datos adquiridos en la exploración y los datos modelados con la ecuación de onda. Sin embargo, para lograr esta imagen del subsuelo se tienen limitaciones computacionales en términos de alto consumo de memoria (terabytes de información) y tiempo de ejecución (semanas o meses de duración).

En el presente proyecto de investigación se busca definir una estrategia de implementación que ayude a lidiar con el costo computacional de la técnica de inversión en el dominio 3D cuando se quieren procesar datos que se consideran de dimensiones masivas (modelos de velocidad y densidad desde 500 x 500 x 140 puntos). La estrategia hace uso de APIs como MPI y OpenMP y la arquitectura CUDA sobre un clúster de cómputo heterogéneo.

Los resultados muestran que es posible aprovechar el gran acceso a memoria del que dispone la CPU mientras se reduce el tiempo de ejecución de las etapas de mayor complejidad computacional, propagación y retro propagación, usando la GPU. Adicionalmente, se encontró que estas etapas de propagación son de naturaleza de algoritmos limitados por memoria (memory bound) y que uso de ancho de banda de la memoria de GPU es una métrica de rendimiento importante. Finalmente, con la estrategia desarrollada se logró procesar alrededor de ocho veces la cantidad de elementos que se logró en un trabajo de investigación previo en el mismo tiempo de ejecución.

---

[*] Trabajo de maestría

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Dr. Ing. Sergio A. Abreo C. Co-Director: Ana B. Ramírez S.

ABSTRACT

TITLE: COMPUTATIONAL IMPLEMENTATION STRATEGY OF THE 3D FULL WAVEFORM INVERSION FOR REAL DATA USING A GPU CLUSTER [*]

AUTHOR: REYNALDO FABIAN NORIEGA ZAMBRANO [**]

KEYWORDS: 3D FULL WAVEFORM INVERSION, FDTD, GPU ARCHITECTURE, MEMORY HIERARCHY.

DESCRIPTION:

One of the main objectives of oil and gas industry is to find high resolution images of the Earth?s subsurface. Full Waveform Inversion (FWI) is an iterative process that estimates subsurface characteristics (e.g. velocity and density) by minimizing the difference between the acquired data and modeled data. However, in time domain, the FWI method takes long execution times (weeks or months) and requires high memory capacity (terabytes of memory space) to store data. These two aspects are critical in a three-dimensional implementation of the method. This project seeks to define a computational strategy to implement the FWI method on a GPU cluster. The main objective is to process real size seismic datasets and evaluate the performance of the strategy by measuring the execution time and RAM consumption. The strategy takes advantage of the main characteristics of the CPU (access to a huge RAM space) and GPU (high level of parallelism to solve high computational complexity tasks) by using OpenMP and MPI APIs and CUDA architecture. The results show that it was possible to process what is considered as real size seismic datasets (models with 500 x 500 x 140 data points or more). Additionally, the implementation show that the forward and backward modeling process are memory bound like algorithms and its performance should be measured in terms of the GPU memory bandwidth utilization. Finally, it was possible to deal with eight times, in the same execution time, the amount of data processed in a previous master?s thesis inside the research group.

---

[*] Master Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Advisor: Dr. Ing. Sergio A. Abreo C. Co-Advisor: Ana B. Ramírez S..

INTRODUCCIÓN

En el sector de hidrocarburos y minería se necesitan métodos para generar imágenes del subsuelo que brindan información para decidir sobre la extracción de elementos de interés como lo pueden ser gases o minerales. En los últimos años, el método de Inversión de Onda Completa (FWI, por sus siglas en inglés)ha sido muy utilizado debido a la capacidad de generar imágenes de alta calidad del subsuelo terrestre (comparado con técnicas como la tomografía). La FWI es un método de optimización local que minimiza la diferencia entre los datos adquiridos en la exploración sísmica con los datos modelados mediante la solución de la ecuación de onda. Teniendo esto en cuenta, una estimación inicial(modelos de velocidades, densidades, etc) se actualiza de manera iterativa hasta cumplir una determinada condición de salida. Sin embargo, la implementación del algoritmo de inversión implica el uso de una compleja infraestructura computacional para lididiar con los problemas de alto consumo de memoria (del orden de terabytes) y tiempo de ejecución (semanas o meses). La computación de alto desempeño (HP C, por sus siglas en inglés) es un área que ha ayudado en la resolución de problemas a gran escala en áreas como ingeniería, medicina, la exploración sísmica, economíá, etc. Debido al alto costo computacional del método F W I, se han necesitado costosos y complejos equipos de cómputo para obtener modelos 3D del subsuelo [?]. Teniendo esto en cuenta, se necesitan encontrar estrategias de manejo de recursos que ayuden a generar el mismo resultado de alta calidad pero reduciendo el consumo de memoria y duración del proceso. El presente trabajo muestra las implicaciones, en términos de consumo de almace- namiento y tiempo de ejecución, de llevar a cabo la F W I en el dominio 3D. Debido a esto, se plantea una estrategia computacional para implementar la técnica de inversión usando un cluster heterogéneo (que usa unidades de procesamiento central y unidades de procesamiento gráfico). La definición de la estrategia

está enfocada en la reducción tanto del consumo de memoria como del impacto al tiempo de ejecución que ésto implica.

1. OBJETIVOS

Objetivo general

- Implementar la técnica de Inversión de Onda Completa (IOC) 3D, isótropa acústica con densidad variable en el dominio del tiempo mediante el esquema de diferencias finitas (FDTD), para datos reales en un clúster heterogéneo.

Objetivos específicos

- Definir una estrategia computacional para implementar la IOC 3D para procesar datos reales sobre un cl ?uster heterog eneo.;
- Implementar la IOC 3D para procesar datos reales sobre el clúster CPS usando la estrategia definida.;
- Validar las imágenes obtenidas con la implementación propuesta usando modelos de velocidad y densidad 3D sintéticos;
- Evaluar el desempeño computacional de la implementación a nivel de tiempo de ejecución, porcentaje de utilización de la GPU y consumo de memoria tanto de GPU como memoria principal del sistema.

2. ECUACIÓN DE ONDA E INVERSIÓN DE ONDA COMPLETA

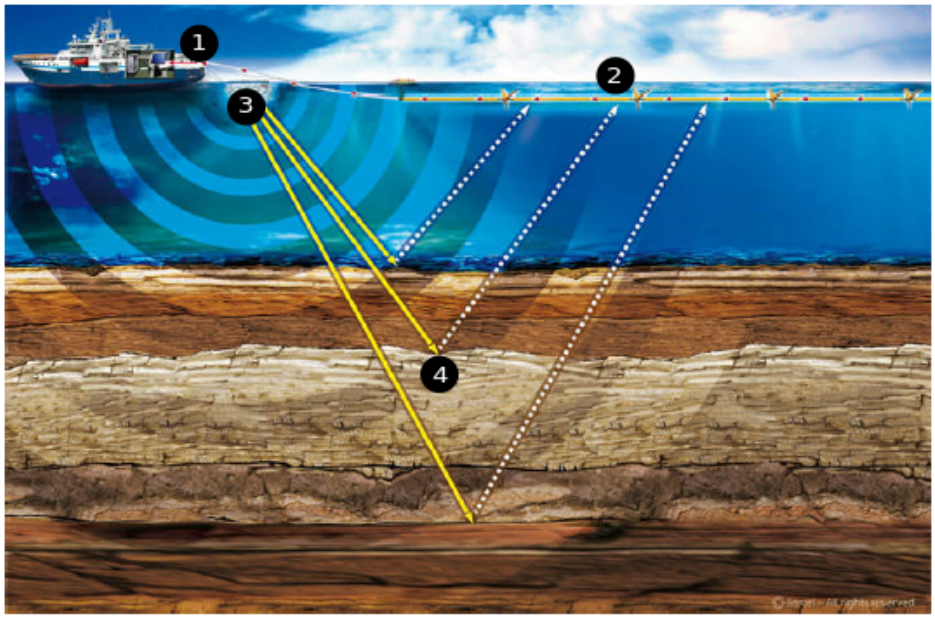
2.1. Exploración sísmica marina

En el sector de hidrocarburos y minería, las industrias de exploración sísmica necesitan métodos para generar imágenes del subsuelo terrestre que permitan tomar decisiones sobre la extracción de elementos de interés como lo son minerales, petróleo, etc. La entrada a estos algoritmos de procesamiento sísmico es el conjunto de información conocido como datos observados o datos adquiridos, siendo un dato marino aquel que proviene de una adquisición marina y un dato terrestre aquel que proviene de una exploración terrestre. En la exploración sísmica marina, el método de adquisición consiste en un barco arrastrando un conjunto de fuente (pistola o cañón de aire) y receptores (hidrófonos) (ver figura 1). Cada vez que se acciona la fuente, los receptores van a grabar la información asociada a la interacción por el campo de presión generado y las diferentes capas del subsuelo.

Si se grafica la información adquirida con los hidrófonos, se tienen diferentes matrices (conjunto de trazas sísmicas o shotgather) con el comportamiento que se aprecia en la figura 2. La cantidad de matrices depende de la cantidad de líneas de receptores (conocidas como streamers) usados en el proceso (hasta un máximo de 16 streamers en las adquisiciones más costosas). Después de adquirir los datos, se necesita una técnica que ayuda a traducir esta información en imágenes del subsuelo para su posterior interpretación.

En el último par de décadas, el método de inversión de onda completa (FWI, por sus siglas en inglés) ha sido muy utilizado gracias a su capacidad de generar imágenes de alta resolución del subsuelo terrestre en comparación con técnicas como la tomografía (ver figura 3). Su uso se ha extendido tanto en la industria como en la academia ya que con el paso del tiempo se han encontrado nuevas áreas de aplica-

Figura 1. Adquisición marina [1]. Un barco que arrastra la fuente (1) y el conjunto de receptores(2) recorre la zona donde se desea adquirir la información mediante la interacción del campo de presión generado por la fuente (3) y las capas del subsuelo(4).



ción (*e.g.* En medicina [4] se utiliza el método de inversión para reconstruir imágenes del cuerpo humano y detectar anomalías como tumores cancerígenos).

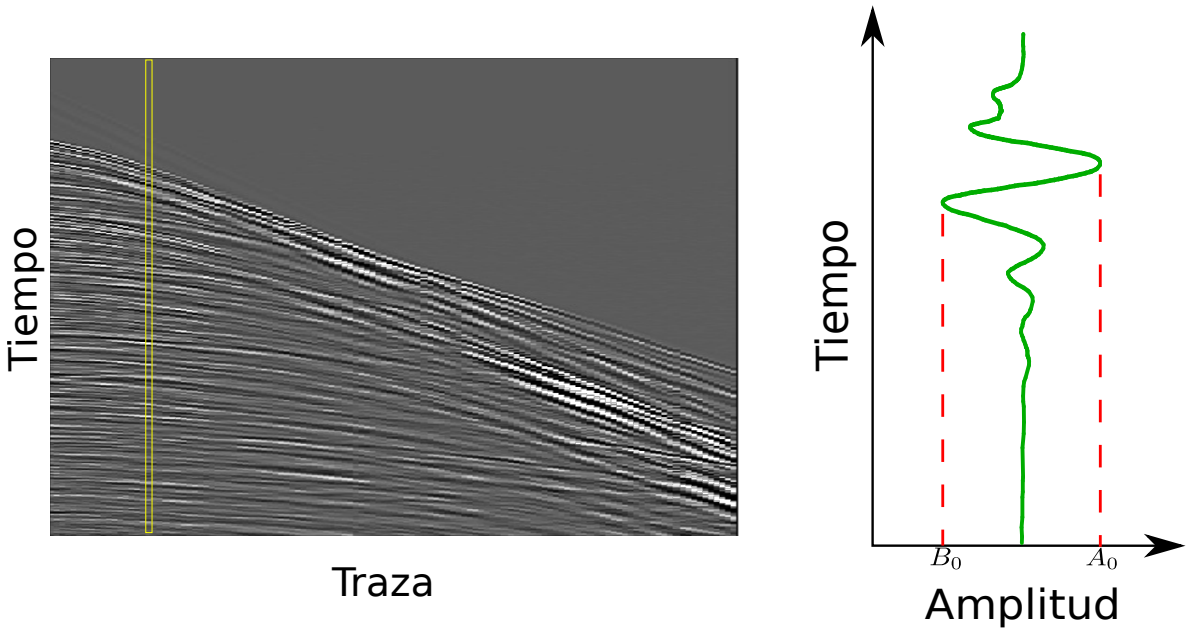
2.2. Ecuación de onda acústica isótropa con densidad variable

El núcleo de operación de métodos como Inversión de Onda Completa y Migración Reversa en el Tiempo es el modelado de la ecuación de onda. Entre más complejo sea el operador usado, más características de los parámetros serán proyectados en los datos modelados y la correlación con los datos adquiridos será mejor. Algunos tipos de ecuación de onda son:

---

[4] Oscar Calderon Agudo y col. "3D imaging of the breast using full-waveform inversion". En: *Proceedings of the International Workshop on Medical Ultrasound Tomography: 1.-3. Nov. 2017, Speyer, Germany*. KIT Scientific Publishing. 2018, pág. 99.

Figura 2. Ejemplo de dato observado o adquirido (Modificado de [2]). Izquierda: Conjunto de trazas en dominio 2D [3]. Derecha: Forma de traza sísmica.

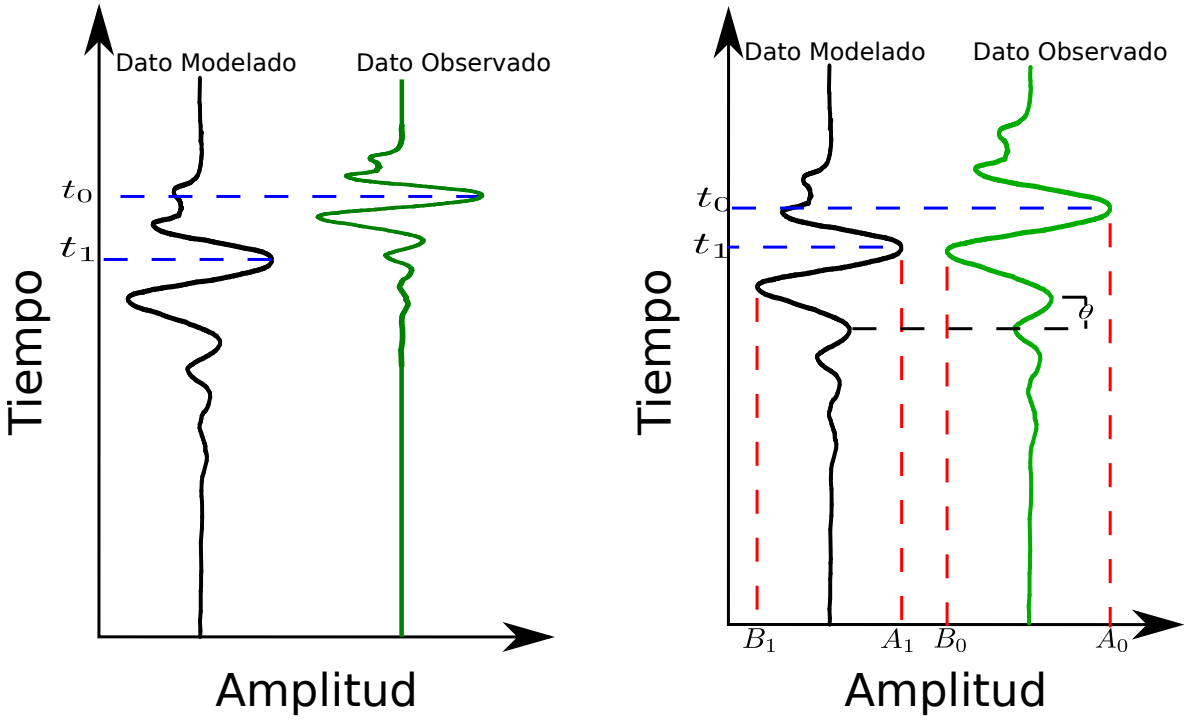


- Ecuación de onda acústica isótropa con densidad constante.
- Ecuación de onda acústica isótropa con densidad variable.
- Ecuación de onda acústica anisótropa densidad variable.
- Ecuación de onda elástica isótropa.
- Ecuación de onda elástica anisótropa.

Debido a la experiencia del grupo de investigación CPS con la ecuación de onda, FWI y datos reales en el dominio 2D [6], el operador usado en este trabajo es la ecuación de onda acústica isótropa con densidad variable, representada por el conjunto

[6] Sergio Abreo y col. "Diving-wave FWI methodology applied to a Colombian Caribbean data set". En: *The Leading Edge* 37.4 (2018), págs. 291-295.

Figura 3. Comparación del comportamiento de la tomografía (izquierda) y la técnica FWI (derecha). Mientras que la tomografía se encarga de ajustar los tiempos t donde ocurren las máximas amplitudes del dato modelado y observado, la FWI ajusta el desfase entre ambos datos y las amplitudes de los mismos. Para que la inversión de onda completa funcione correctamente es necesario que el máximo desfase en tiempo entre los datos observados y modelados no supere más de la mitad del periodo de la ondícula central. [5]



de ecuaciones (1), (2), (3) y (4)

$$\frac{1}{\rho \mathbf{c^2}} \frac{\partial \mathbf{P}}{\partial t} = -\frac{\partial \mathbf{V_x}}{\partial x} - \frac{\partial \mathbf{V_y}}{\partial y} - \frac{\partial \mathbf{V_z}}{\partial z} \tag{1}$$

$$\rho \frac{\partial \mathbf{V_x}}{\partial t} = -\frac{\partial \mathbf{P}}{\partial x} \tag{2}$$

$$\rho \frac{\partial \mathbf{V_y}}{\partial t} = -\frac{\partial \mathbf{P}}{\partial y} \tag{3}$$

$$\rho \frac{\partial \mathbf{V_z}}{\partial t} = -\frac{\partial \mathbf{P}}{\partial z} \tag{4}$$

donde ρ y $\mathbf{c}$ son los parámetros de densidad y velocidad, respectivamente, x, y y z son las tres dimensiones del dominio espacial, t es la variable que representa el tiempo, $\mathbf{P} \in \mathbb{R}^{N_{xyzt}} (N_{xyzt} = N_x \times N_y \times N_z \times N_t)$ es el campo de presión y $\mathbf{V_x} \in \mathbb{R}^{N_{xyzt}}$, $\mathbf{V_y} \in \mathbb{R}^{N_{xyzt}}$ y $\mathbf{V_z} \in \mathbb{R}^{N_{xyzt}}$ son los desplazamientos de velocidad o $momentum$ en los ejes x, y y z, respectivamente.

El operador definido se implementa usando diferencias finitas en el dominio del tiempo (FDTD) con una aproximación de octavo orden para las derivadas espaciales y una aproximación de primer orden para las derivadas temporales [7] teniendo en cuenta el uso de malla intercalada [8]. De esta forma, los campos $\mathbf{P}$, $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$ (para su discretización temporal) se escriben como aparece en la expresión (5)

$$\frac{\partial \mathbf{U}}{\partial t} \approx \frac{\mathbb{U}^{n+1}_{i,j,k} - \mathbb{U}^{n}_{i,j,k}}{\delta t}, \tag{5}$$

donde $\mathbf{U}^n$ es cualquiera de estos campos discretos; i,j,k y n representan las variables discretas para x, y, z y t, respectivamente, y δt es el paso temporal. Puede verse como una expresión en términos de los campos en el instante de tiempo presente ($\mathbb{U}^{n}_{i,j,k}$) y futuro ($\mathbb{U}^{n+1}_{i,j,k}$).

Para las aproximaciones espaciales, el campo $\mathbf{P}$ se escribe como sigue (ver figura 4):

$$\frac{\partial \mathbf{P}}{\partial x} \approx \sum_{l=0}^{L-1} \frac{\beta_l \cdot (\mathbb{U}^{n}_{i+l+1,j,k} - \mathbb{U}^{n}_{i-l,j,k})}{\delta x}, \tag{6}$$

---

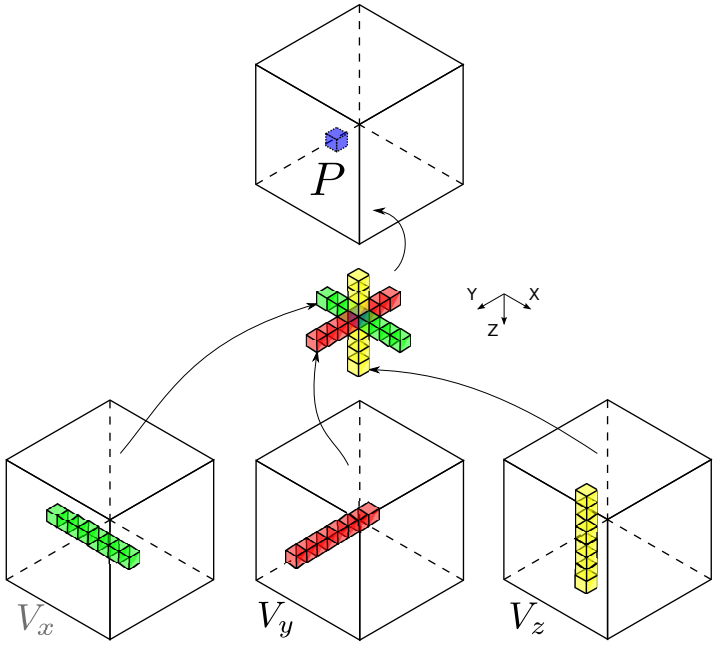
[7] Dennis M Sullivan. *Electromagnetic simulation using the FDTD method*. John Wiley & Sons, 2013.

[8] Jean Virieux y Raul Madariaga. "Dynamic faulting studied by a finite difference method". En: *Bulletin of the Seismological Society of America* 72.2 (1982), págs. 345-369.

$$\frac{\partial \mathbf{P}}{\partial y} \approx \sum_{l=0}^{L-1} \frac{\beta_l \cdot (\mathbb{U}^n_{i,j+l+1,k} - \mathbb{U}^n_{i,j-l,k})}{\delta y}, \tag{7}$$

$$\frac{\partial \mathbf{P}}{\partial z} \approx \sum_{l=0}^{L-1} \frac{\beta_l \cdot (\mathbb{U}^n_{i,j,k+l+1} - \mathbb{U}^n_{i,j,k-l})}{\delta z}, \tag{8}$$

Figura 4. Cálculo de una muestra de tiempo del campo $\mathbf{P}$ según el conjunto de ecuaciones 6, 7 y 8. Nótese que para calcular cada elemento de $\mathbf{P}$ se necesitan ocho elementos de los cubos $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$.



Adicionalmente, los campos $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$ se aproximan como las expresiones (9), (10) y (11), respectivamente (ver figura 5).

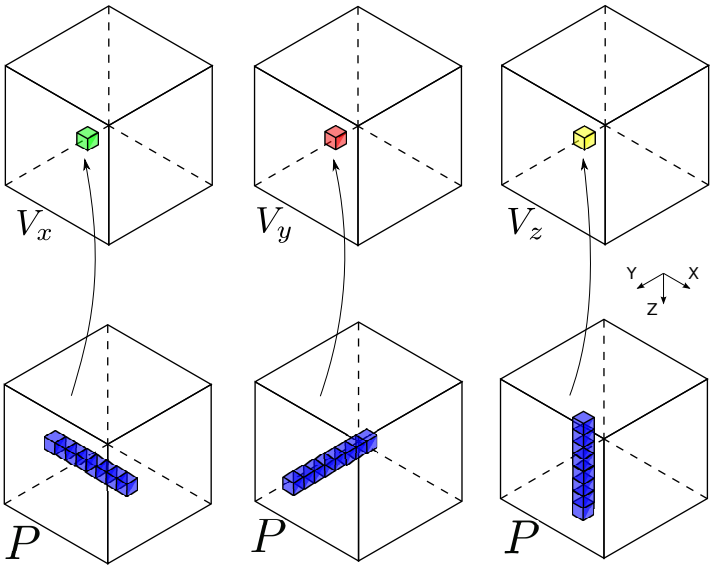
$$\frac{\partial \mathbf{V_x}}{\partial x} \approx \sum_{l=0}^{L-1} \frac{\beta_l \cdot (\mathbb{U}^n_{i+l,j,k} - \mathbb{U}^n_{i-l-1,j,k})}{\delta x}, \tag{9}$$

$$\frac{\partial \mathbf{V_y}}{\partial y} \approx \sum_{l=0}^{L-1} \frac{\beta_l \cdot (\mathbb{U}^n_{i,j+l,k} - \mathbb{U}^n_{i,j-l-1,k})}{\delta y}, \tag{10}$$

$$\frac{\partial \mathbf{V_z}}{\partial z} \approx \sum_{l=0}^{L-1} \frac{\beta_l \cdot (\mathbb{U}^n_{i,j+l,k} - \mathbb{U}^n_{i,j,k-l-1})}{\delta z}, \tag{11}$$

donde β es un arreglo que contiene los coeficientes de diferencias finitas para la malla intercalada, $\beta = [\frac{1225}{1024}, \frac{-245}{3072}, \frac{49}{5120}, \frac{-5}{7168}]$, L es la cantidad de coeficientes, δx, δy y δz representan la resolución espacial del modelo.

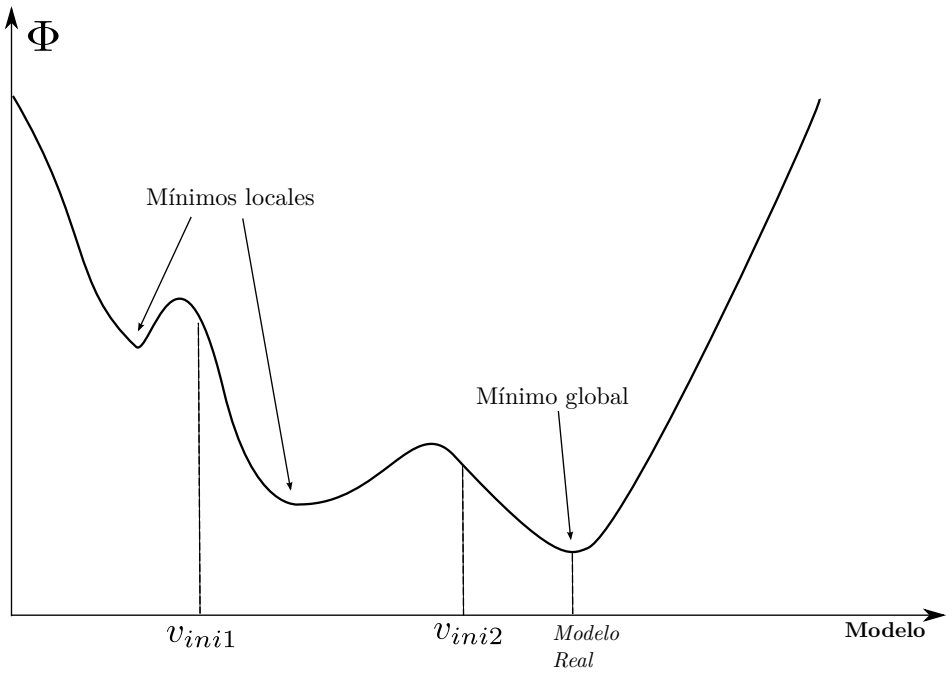
Figura 5. Cálculo de una muestra de tiempo de los campos $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$ según el conjunto de ecuaciones 9, 10 y 11. Nótese que para calcular un elemento de cada uno de estos tres cubos se necesitan ocho elementos del campo $\mathbf{P}$.



2.3. Inversión de Onda Completa en el dominio del tiempo

Debido a que el método de inversión es un problema mal puesto, se pueden obtener infinitas soluciones que generan el mismo conjunto de datos. Esta no unicidad es generada por la existencia de mínimos locales en la función de costo del método, donde cada punto de ésta (figura 6) corresponde a un conjunto de parámetros (densidad y velocidad para este caso). Los parámetros verdaderos son los que generan el valor del mínimo global en esta función de costo.

Figura 6. Mínimos locales y mínimo global en una función de costo determinada de la *FWI*. Partiendo desde v_{ini2} es más probable converger hacia el mínimo global en lugar de usar v_{ini1} como punto inicial. (Modificado de [9]).



La técnica FWI requiere del uso de la ecuación de onda, una función de costo $\Phi(\hat{\mathbf{m}})$ y un método de actualización de parámetros para estimar el modelo sísmico. Usualmente, la función de costo se define como la curvatura formada por los errores entre el dato modelado y el dato adquirido (datos residuales)en cada iteración de la FWI [10]. Para esta técnica se utiliza la norma ℓ_2 de los datos o trazas residuales tal como se muestra en la ecuación (12)

$$\Phi(\hat{\mathbf{m}}) = \arg\min_{\hat{\mathbf{m}}} ||d_{mod}(\hat{\mathbf{m}}) - d_{obs}(\mathbf{m})||_2^2 \,, \tag{12}$$

donde los datos modelados d_{mod} se calculan usando la ecuación de onda, $\hat{\mathbf{m}}$ es

[10] A Tarantola. "Inversion of seismic-reflection data in the acoustic approximation". En: *Geophysics* 48.8 (1984), págs. 1259-1266.

el conjunto de parámetros obtenidos con FWI y $\mathbf{m}$ es el conjunto de parámetros verdaderos.

De manera iterativa, la actualización de los parámetros se realiza mediante expansión de la serie de Taylor alrededor de un punto, tal como se ve en la expresión (13)

$$\hat{\mathbf{m}}^{k+1} = \hat{\mathbf{m}}^{k} + \alpha_k(-\left(\mathbf{H}(\hat{\mathbf{m}}^{k})\right)^{-1}\mathbf{g}(\hat{\mathbf{m}}^{k}))\ , \tag{13}$$

donde α_k es la magnitud de avance de la dirección, hacia el mínimo global, indicada por el gradiente ($\mathbf{g}(\hat{\mathbf{m}}^{k})$) [11] y la inversa de $\mathbf{H}(\hat{\mathbf{m}}^{k})$ (la Hessiana) indica la curvatura de la función de costo alrededor de ese punto.

Existe una metodología que permite obtener el adjunto del operador de onda escogido [12] y que depende de cómo éste se defina. Adicionalmente, con esta metodología se pueden obtener los gradientes con los que se van a actualizar los parámetros dentro del método de inversión. Esta metodología fue interpretada y documentada por un estudiante de maestría de la universidad tecnológica de Delft, Holanda, en intercambio con el grupo de investigación CPS.

Para el operador de onda seleccionado en este trabajo, el adjunto está dado por el conjunto de expresiones (14), (15), (16) y (17)

$$\frac{1}{\rho\mathbf{c^2}}\frac{\partial\lambda}{\partial t} = -\frac{\partial\lambda_{\mathbf{x}}}{\partial x} - \frac{\partial\lambda_{\mathbf{y}}}{\partial y} - \frac{\partial\lambda_{\mathbf{z}}}{\partial z} \tag{14}$$

---

[11] J. Nocedal y S. J. Wright. *Numerical Optimization*. 2nd. New York: Springer, 2006.

[12] Jaap J. Wesdorp. *A general recipe for obtaining adjoint equations and gradients and its application to full waveform inversion of 2D isotropic elastic media in finite difference time-domain*. Inf. téc. Master internship report. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones, Universidad Industrial de Santander, 2018.

$$\rho \frac{\partial \lambda_{\mathbf{x}}}{\partial t} = -\frac{\partial \lambda}{\partial x} \tag{15}$$

$$\rho \frac{\partial \lambda_{\mathbf{y}}}{\partial t} = -\frac{\partial \lambda}{\partial y} \tag{16}$$

$$\rho \frac{\partial \lambda_{\mathbf{z}}}{\partial t} = -\frac{\partial \lambda}{\partial z} \tag{17}$$

donde $\lambda \in \mathbb{R}^{N_{xyzt}}$ es el campo de presión generado por la información de residuales (diferencia del dato observado y el dato modelado) y los campos $\lambda_{\mathbf{x}} \in \mathbb{R}^{N_{xyzt}}$, $\lambda_{\mathbf{y}} \in \mathbb{R}^{N_{xyzt}}$ y $\lambda_{\mathbf{z}} \in \mathbb{R}^{N_{xyzt}}$ son los desplazamientos en los ejes x, y y z, respectivamente, teniendo un operador autoadjunto para este caso.

Para cada cada parámetro que se está invirtiendo se obtiene un gradiente diferente. Para actualizar la velocidad y la densidad se utilizan los gradientes calculados con las ecuaciones (18) y (19), respectivamente.

$$G_{\mathbf{c}} = \frac{-2}{\rho \mathbf{c}^3} \int_0^T \frac{\partial \mathbf{P}}{\partial t} \cdot \lambda \; \delta t \tag{18}$$

$$G_{\rho} = \int_0^T \frac{-1}{\rho^2 \mathbf{c}^2} \frac{\partial \mathbf{P}}{\partial t} \cdot \lambda + \frac{\partial \mathbf{V_x}}{\partial t} \cdot \lambda_x + \frac{\partial \mathbf{V_y}}{\partial t} \cdot \lambda_y + \frac{\partial \mathbf{V_z}}{\partial t} \cdot \lambda_z \; \delta t \tag{19}$$

Por otro lado, la Hessiana no se calcula generalmente debido a su alto costo computacional (complejidad computacional $O(N^5)$). Por esta razón, la actualización de los parámetros se realiza usualmente como se aprecia en la expresión (20)

$$\hat{\mathbf{m}}^{k+1} = \hat{\mathbf{m}}^k + \alpha_k \mathbf{h}(\hat{\mathbf{m}}^k) \; , \tag{20}$$

donde $\mathbf{h}(\hat{\mathbf{m}}^k)$ es la dirección de búsqueda, la cuál puede ser obtenida por métodos

como el gradiente descendente o L-BFGS [13].

2.4. Costo computacional de la FWI 3D

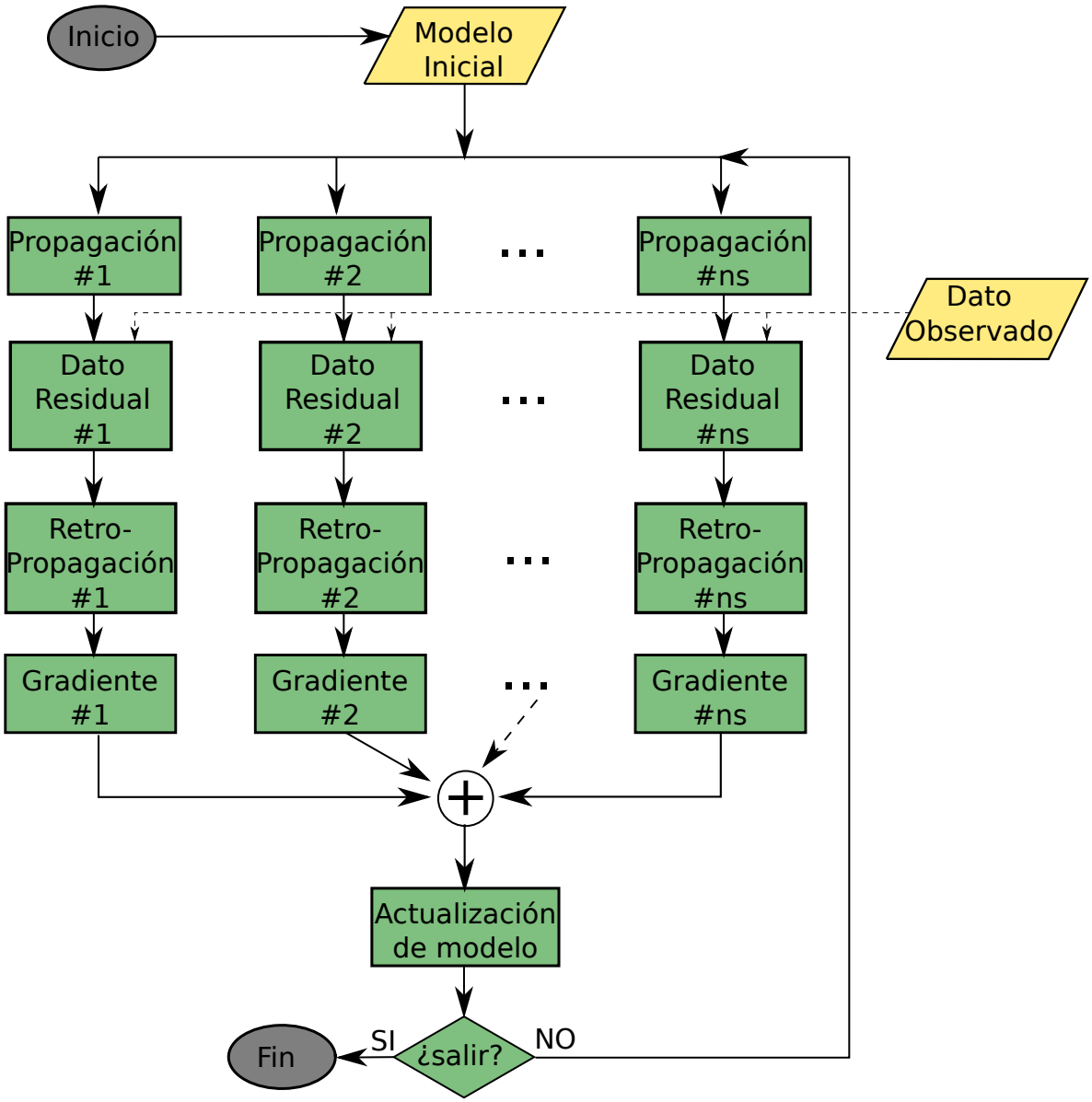
Ya se mencionó que el núcleo de operación del método de inversión es el modelado de la ecuación de onda. Es común referirse al modelado del operador de onda y su adjunto como propagación y retro-propagación, respectivamente. Estas dos etapas dentro del proceso de inversión son las que demandan mayor tiempo de ejecución y capacidad de memoria (complejidad computacional $O(N^4)$). En la figura 7 se muestra el esquema general del algoritmo FWI, donde en cada iteración del método se requiere hacer tanto una propagación como una retropropagación por cada fuente que se esté simulando. La ventaja de este esquema es que no se necesitan calcular los gradientes en un determinado orden por lo que son operaciones independientes. La característica importante es que el gradiente de actualización de los parámetros de inversión es la acumulación de la información aportada por todas las fuentes.
De acuerdo a las ecuaciones (18) y (19) se necesita la información de ocho hipercubos de datos ($\mathbf{P}$, $\mathbf{V_x}$, $\mathbf{V_y}$, $\mathbf{V_z}$, λ, $\lambda_\mathbf{x}$, $\lambda_\mathbf{y}$ y $\lambda_\mathbf{z}$) para generar los gradientes de velocidad y densidad en cada iteración de la FWI. Cada hipercubo de datos tiene un comportamiento similar al que se presenta en la figura figura 8 (campo de presión $\mathbf{P}$). Si se quiere procesar datos de dimensiones reales[14], se requieren terabytes de capacidad de almacenamiento para guardar la información asociada a todos estos campos. Para un experimento que maneje volúmenes de velocidad y densidad de dimensiones $500 \times 500 \times 140$ y una fuente con 4500 muestras de tiempo se nece-

---

[13] D. C. Liu y J. Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization". En: *Math. Program.* 45.3 (1989), págs. 503-528. DOI: 10.1007/BF01589116.

[14] Es común referirse como datos masivos o de talla real a modelos sísmicos de dimensiones desde $500 \times 500 \times 140$ [puntos] (dimensiones bastante comunes en la industria) (https://wiki.seg.org/wiki/. Accedido por última vez: 2018-11-28).

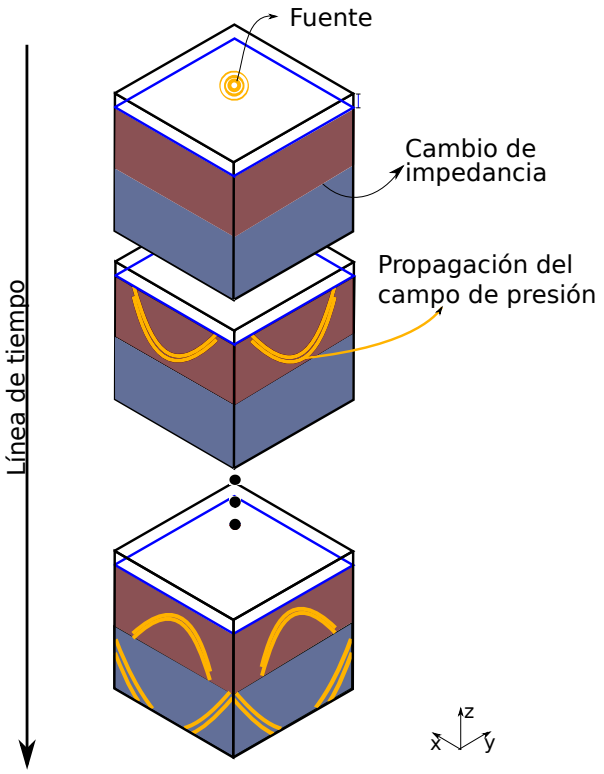
Figura 7. Diagrama de flujo del método FWI. Se realiza el cálculo de un gradiente por cada fuente que se disponga en el proceso (con un total de ns). Se realizan tantas iteraciones como indique la condición de salida (un umbral en la función de costo, una cantidad fija de iteraciones, etc.).



sitan $5.04[TB]$ de capacidad de memoria para almacenar los ocho hipercubos de datos. Teniendo en cuenta que el tiempo de acceso para escritura o lectura de un

disco duro mecánico (con la suficiente capacidad de almacenamiento) es de un par de decenas de millones de ciclos de reloj de procesador, este proceso podría demandar meses o incluso años de tiempo de ejecución para obtener una imagen aceptable del subsuelo terrestre.

Figura 8. Hipercubo que representa la evolución del campo de presión $\mathbf{P}$ durante la etapa de propagación. Cada muestra de este volumen tiene las dimensiones de los modelos de velocidad y densidad que se están procesando ($N_x \times N_y \times N_z$) y hay tantos cubos como muestras tenga la fuente. Los hipercubos de los campos $\mathbf{V_x}$, $\mathbf{V_y}$, $\mathbf{V_z}$, λ, $\lambda_\mathbf{x}$, $\lambda_\mathbf{y}$ y $\lambda_\mathbf{z}$ tendrán las mismas dimensiones aquí mostradas.



3. COMPUTACIÓN DEL ALTO RENDIMIENTO

La computación de alto rendimiento (HPC, por sus siglas en inglés) se puede definir como la práctica de agregar potencia computacional de tal forma que se puede extraer un alto rendimiento de las aplicaciones enfocadas a resolver problemas en las áreas de ingeniería, ciencias, negocios, etc.

Los dos componentes de una plataforma HPC son las aplicaciones (software) diseñadas para ejecutarse sobre el equipo de cómputo y el conjunto de dispositivos (hardware) sobre el cuál se ejecutan estas aplicaciones. Este hardware es generalmente la interconexión de múltiples nodos de cómputo (el equivalente a un equipo de computo de escritorio pero con procesadores y dispositivos de alta gama diseñados para cómputo[15]) que conforman un rack y la conexión de racks es un clúster de cómputo (ver figura 9). Los supercomputadores a nivel mundial [16] están constituidos por decenas o centenares de estos racks de cómputo.

Un campo importante en HPC es el paradigma de la programación paralela, el cuál permite utilizar equipos de cómputo con centenares de unidades de procesamiento central (CPUs, por sus siglas en inglés) y unidades de procesamiento gráfico (GPUs, por sus siglas en inglés). La programación paralela permite realizar operaciones sobre arreglos de datos de largas dimensiones, donde cada elemento se opera de forma independiente, pero al mismo tiempo, usando cientos de núcleos de CPU o de GPU (ver figuras 10 y 11).

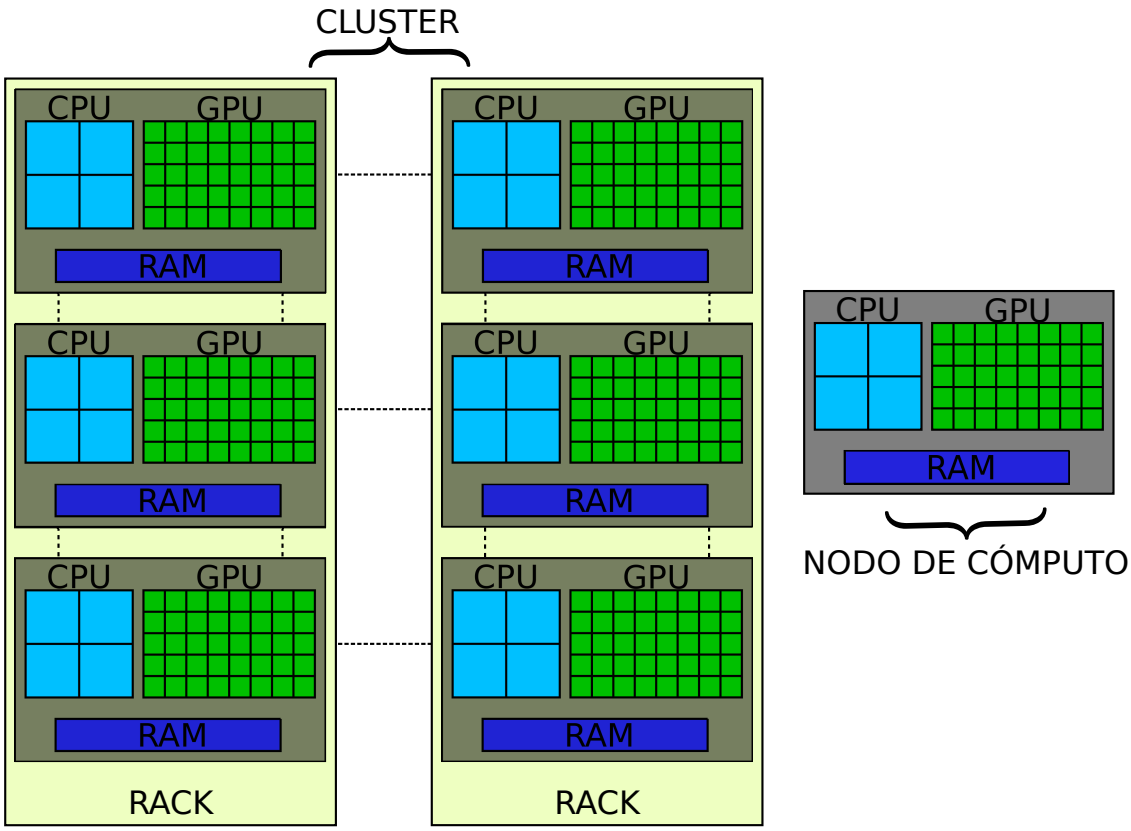
De la mano del paradigma de la programación en paralelo están los lenguajes de programación que permiten implementar aplicaciones HPC. Dependiendo del dispo-

---

[15] Intel Xeon, AMD Epyc, Nvidia Tesla, AMD Radeon Instinct.

[16] `https://www.top500.org/lists/2017/11/`. Accedido por última vez: 2018-11-28.

Figura 9. Diagrama de clúster de cómputo de alto rendimiento. La unidad más básica es el nodo de cómputo que contiene hardware especializado como GPUs y CPUs de alta gama junto con un par de centenares de gigabytes de memoria principal (la memoria de accesso aleatorio o RAM, por sus siglas en inglés).



sitivo con el que se desea trabajar (CPU o GPU) se elije el entorno de programación. Algunos lenguajes o APIs(Application Programming Interface) para programación en paralelo, tanto de GPUs(verde) como de CPUs (azul), son:

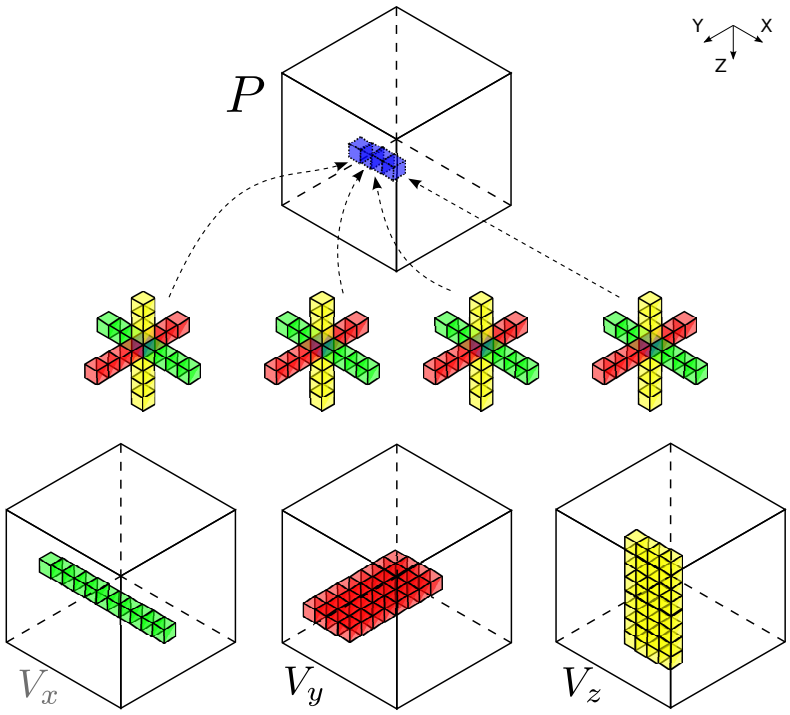
- PThreads [17]
- OpenMP [18]

[17] https://computing.llnl.gov/tutorials/pthreads/. Accedido por última vez: 2018-11-28.

[18] https://www.openmp.org/. Accedido por última vez: 2018-11-28.

[19] https://www.open-mpi.org/. Accedido por última vez: 2018-11-28.

Figura 10. Ejemplo del cálculo de una muestra de tiempo del campo $\mathbf{P}$ utilizando programación paralela (en comparación con el proceso mostrado en la figura 4). Las flechas a trazos representan cuatro hilos de procesamiento diferentes (threads). Cada hilo procesa los datos necesarios para generar, en conjunto, cuatro elementos de una muestra de $\mathbf{P}$ a la vez.



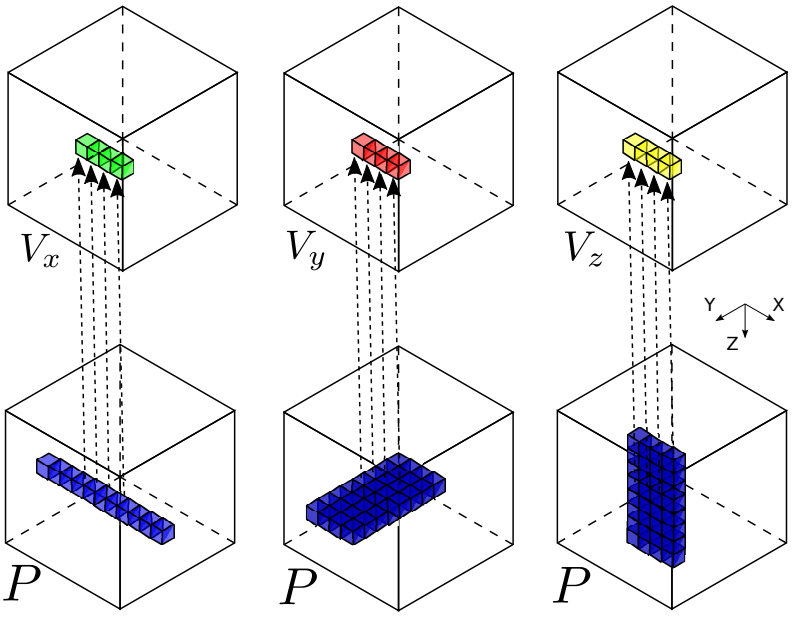
- MPI [19]
- CUDA [20]
- OpenCL [21]
- OpenACC [22]

[20] https://developer.nvidia.com/cuda-zone. Accedido por última vez: 2018-11-28.

[21] https://www.khronos.org/opencl. Accedido por última vez: 2018-11-28.

[22] https://www.openacc.org/. Accedido por última vez: 2018-11-28.

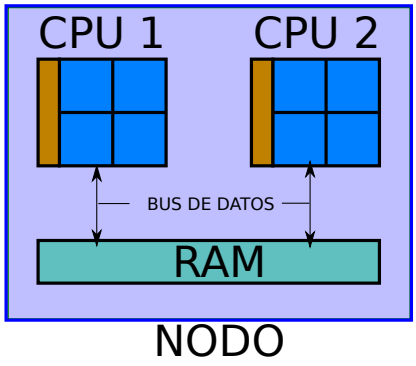
Figura 11. Ejemplo del cálculo de una muestra de tiempo de los campos $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$ usando programación en paralelo (en comparación con el proceso mostrado en la figura 5). Cuatro hilos de procesamiento calculan, al mismo tiempo, cuatro elementos de estos campos a partir de la información de $\mathbf{P}$. Esta operación sería n veces más rápida en comparación con el proceso secuencial si se usan n hilos de procesamiento.



3.1. OpenMP

Del inglés Open Multi Processing, OpenMP es una interfaz de programación para sistemas de memoria compartida diseñada para la implementación de aplicaciones que van a hacer uso de todos los recursos de CPU y memoria al interior de un nodo de cómputo. De esta forma, la ejecución de los algoritmos está limitada por la cantidad de RAM dentro del nodo y la cantidad de núcleos disponibles en el o los procesadores que se comunican con esta memoria (ver figura 12).

Figura 12. Arquitectura de un sistema de memoria compartida sobre la cuál se enfocan las aplicaciones de OpenMP. Los hilos de procesamiento ejecutados desde cualquier CPU acceden y comparten los mismos arreglos de datos almacenados en RAM.



La ventaja de usar OpenMP es su sintaxis de programación sencilla. Generalmente es necesaria una única instrucción o pragma de OpenMP para explotar el posible paralelismo al interior de un bucle que opera sobre arreglos de datos (ver lista 3.1). El factor de aceleración de la aplicación sobre la versión serial depende de la cantidad de núcleos de CPU usados.

Listing 3.1. Ejemplo del uso de un pragma de OpenMP para realizar un bucle for en paralelo. Sólo es necesario agregar la línea que contiene el pragma de OpenMP para dejar de usar un único núcleo de CPU y empezar a utilizar todos los núcleos disponibles.

```
//Algunas tareas en forma serial
...
//Inicio de region en paralelo
#pragma omp parallel for
for(i=0;i<n;i++){
  C[i] = A[i] + B[i];
}
//Fin de region en paralelo
...
//Otras tareas en forma serial
```

3.2. MPI

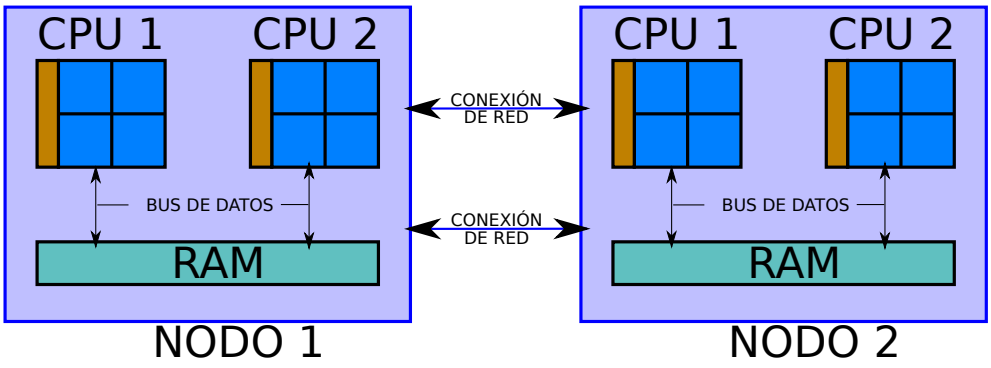
Del inglés Message Passing Interface, MPI es un estándar de programación para sistemas de memoria distribuída en el que se aprovecha la interconexión de los diferentes nodos de cómputo de un clúster para implementar las aplicaciones. Aunque la interfaz permite ejecutar múltiples $threads$ dentro de un mismo nodo de cómputo, la ventaja de su sintaxis es trabajar a nivel de clúster (ver figura 13). Teniendo en cuenta que ya existen sistemas especializados de conexión de red (e.g. InfiniBand [23]), la transferencia de datos entre nodos no es un problema al momento de ejecutar aplicaciones que usan MPI.

Al contrario que OpenMP, su sintaxis es más compleja debido a las instrucciones

[23] `https://www.infinibandta.org/`. Accedido por última vez: 2018-11-28.

utilizadas para la creación de $threads$, el procesamiento y su comunicación entre los diferentes nodos. Adicionalemente, en HPC es bastante común ver procesos que se benefician del uso de MPI y OpenMP a la vez para sacar el máximo provecho del clúster de procesamiento usando las ventajas de cada ambiente de programación. Otra característica de usar MPI es la creación del mismo arreglo de datos tantas veces como hilos de MPI se deseen utilizar.

Figura 13. Arquitectura de un sistema de memoria distribuída para trabajar con MPI. La ventaja de esta interfaz de programación es la comunicación entre diferentes nodos mediante una conexión de red cuya velocidad hoy en día no es un problema gracias a herramientas como InfiniBand.



Si se modifica el esquema del método de inversión mostrado en la figura 7 y se aprovecha la cualidad de MPI para trabajar con múltiples nodos se tiene el diagrama que aparece en la figura 14. Suponiendo que se necesita un nodo completo para generar un gradiente por fuente, si se tienen tantos nodos como cantidad de fuentes a usar, el tiempo necesario para actualizar los parámetros a invertir es el tiempo requerido para generar un solo gradiente asociado a una fuente. Si hay menor cantidad de nodos, cada uno de estos procesará una determinada cantidad de fuentes y esto indicará el factor de aceleración respecto al trabajo con una sola unidad de cómputo.

3.3. CUDA

Del inglés Compute Unified Device Architecture, CUDA es la arquitectura de cómputo diseñada por NVIDIA para trabajar el paradigma de la programación paralela sobre sus GPUs. Dentro del entorno de CUDA existe el lenguaje de programación CUDA-C que consiste en una serie de extensiones del lenguaje C con librerías expecíficas para sacar provecho de la arquitectura de las GPUs. Desde el lanzamiento de la primer GPU, compatible con CUDA, en 2007, la computación de alto desempeño ha crecido exponencialmente gracias tanto al rendimiento que ofrecen estos dispositivos como a la facilidad de escribir software con CUDA-C (en comparación con el lenguaje de programación OpenCL). Debido a esto, el uso de CPUs empieza a enfocarse en otras tareas como administrar procesos de menor complejidad computacional o manejar las transferencias de memoria dentro de cualquier implementación de software.

A diferencia de lo que se conoce como funciones de C, en CUDA-C se implementan kernels (funciones que se ejecutan sobre la GPU). Al esquema de trabajo CPU + GPU se le conoce como computación heterogénea [24], donde se aprovechan las características de más de un tipo de procesador. Un ejemplo de su uso es el que se aprecia en la figura 15 donde la CPU y la GPU operan sobre bloques de datos a través del tiempo. Dentro del entorno de CUDA, es común referirse a la GPU como device y a la CPU como host.

Debido a que cada vez es más considerado el uso de GPUs en los supercomputadores a nivel mundial, las implementaciones de software generalmente hacen uso de hilos de MPI para lanzar tareas en diferentes nodos de cómputo dentro de un mismo cluster. Cada uno de estos hilos se encarga usualmente de administrar tanto las funciones que se lanzan en la CPU como los kernels invocados en la GPU (ver

---

[24] `https://www.linuxjournal.com/article/8368`. Accedido por última vez: 2018-11-28.

figura 16).

Una de las líneas de investigación del grupo CPS es la de computación de alto rendimiento. Para la exploración de esta área de trabajo, el grupo dispone de un clúster que tiene nodos de cómputo con CPUs de la marca Intel y dos GPUs de la marca NVIDIA, los cuáles han permitido desarrollar aplicaciones en paralelo tanto con el uso de CPUs como de GPUs. El lenguaje de programación OpenMP se escogió debido a tanto a su facilidad de programación para ejecutar tareas en paralelo como a su amplio uso en el estado-del-arte. MPI también es otro reconocido API de programación y adicionalemente permitirá la comunicación de tareas entre los tres nodos del clúster disponible. Por último, aunque las GPUs pueden ejecutar código escrito en OpenCL (lenguaje también basado en C pero con una sintaxis muy compleja) y OpenACC (la versión para GPUs de OpenMP) se escoge CUDA debido a que es diseñado también por NVIDIA y se puede extraer mayor rendimiento en comparación con los otros dos lenguajes[26]. Adicionalemente, $CUDA$ tiene una sintaxis menos compleja que OpenCL.

[26] Tetsuya Hoshino y col. “CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application”. En: *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE. 2013, págs. 136-143.

Figura 14. Modificación al diagrama de flujo del método FWI usando MPI. Debido a que la información del gradiente generado por cada fuente se puede calcular en cualquier orden, se puede usar MPI para generar un gradiente en cada nodo que se disponga dentro del clúster donde se esté trabajando. De esta forma se reduce el tiempo de ejecución al calcular n gradientes a la vez, siendo n el número de nodos en el clúster de cómputo.

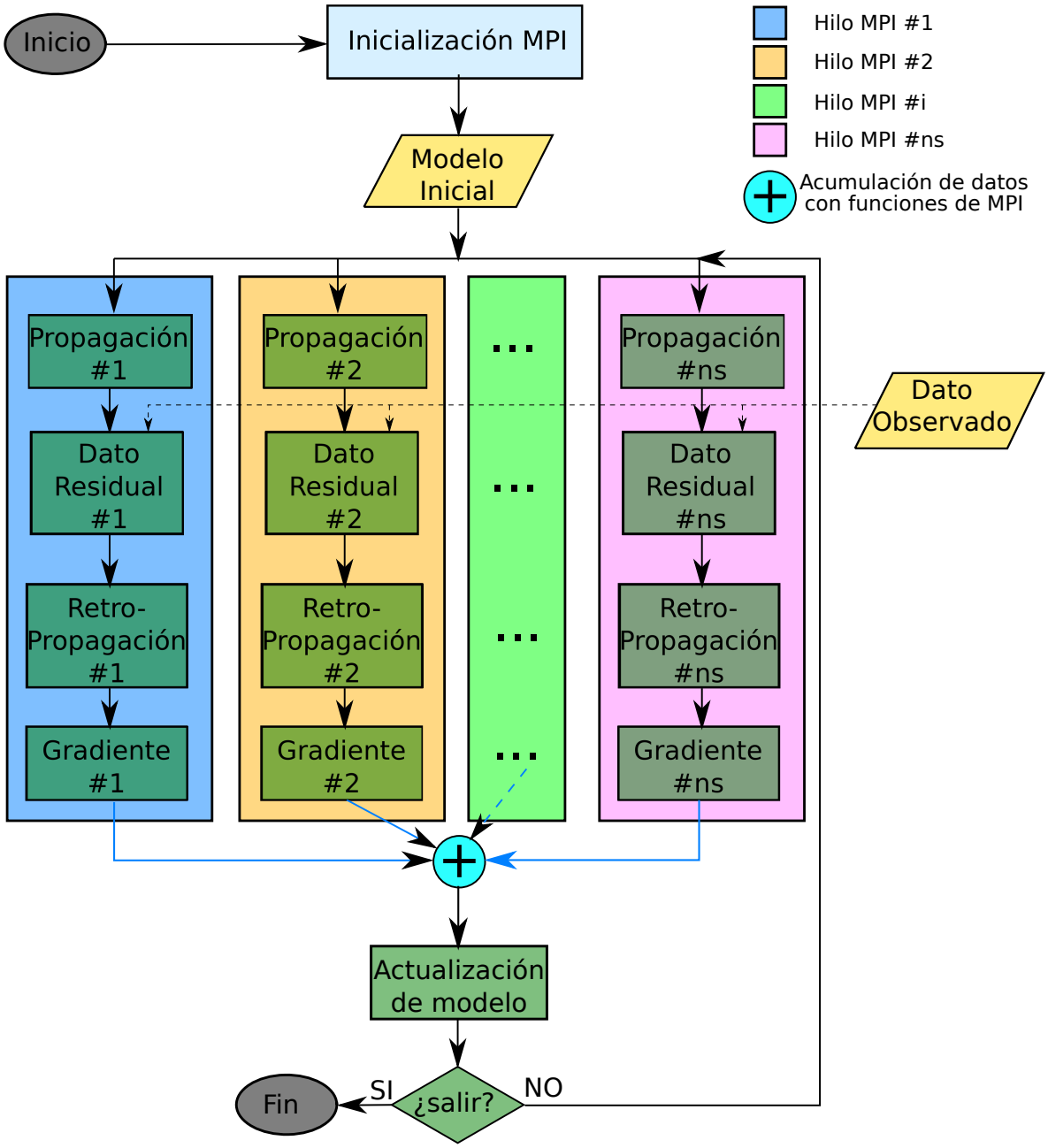


Figura 15. Esquema de programación de cómputo heterogéneo. Los procesos netamente seriales los realiza la CPU y la GPU se encarga de aquellas tareas que se puedan realizar en paralelo. Así mismo, mientras la GPU permanece ocupada, la CPU puede ejecutar otras actividades que no dependan de la información almacenada en la memoria del $device$. (Adaptado de [25]).

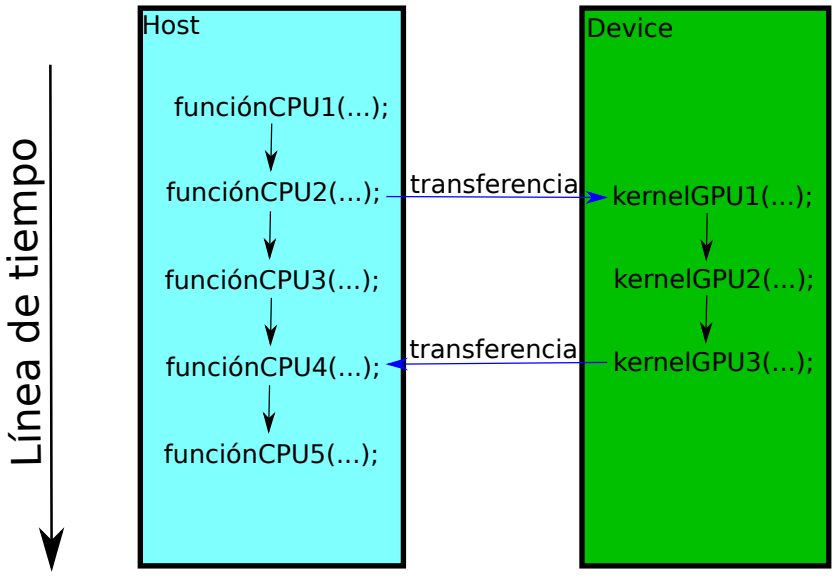
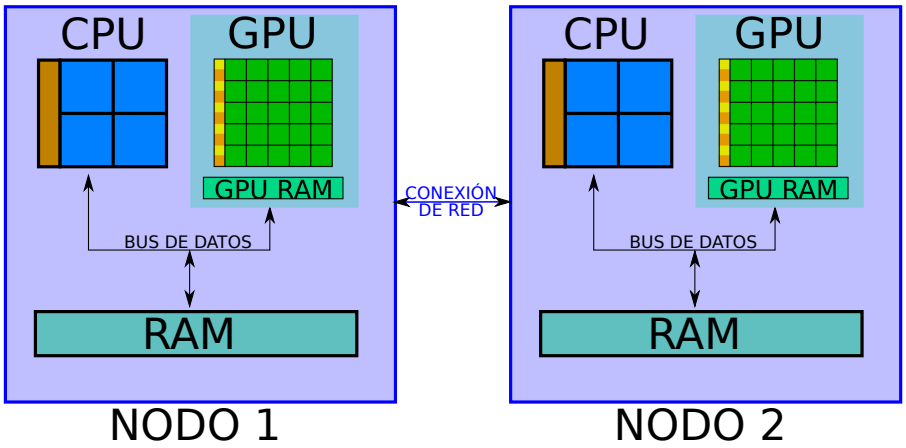


Figura 16. Diagrama de un sistema que usa tanto GPUs como CPUs para resolver problemas en HPC. Si se lanzan dos hilos de MPI cada uno va a usar una pareja CPU y GPU para trabajar sobre un segmento de los datos almacenados en la RAM principal del sistema. Cada thread de MPI se encarga de invocar tanto funciones como kernels de CUDA para aprovechar las ventajas de cada procesador.



4. ESTADO-DEL-ARTE

La FWI demanda un alto costo computacional en términos de memoria y tiempo de ejecución (complejidad computacional $O(N^4)$ para el caso $3D$). Para definir la metodología de implementación sobre una plataforma HPC es necesario realizar un análisis general de las características y/o limitaciones de la arquitectura con la que se va a trabajar (para diseñar software es necesario conocer el hardware) y de las estrategias computacionales en el estado-del-arte para el uso de clusters de cómputo y manejo de memoria.

4.1. Cluster de cómputo Tokyo

El grupo de investigación CPS dispone de un equipo de cómputo compuesto de tres nodos con las características que se muestran a continuación
El primer nodo, cuyo nombre de host es $Tokyo\_00$ contiene:

- CPU: 2x Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz,
- GPU: 2x Nvidia Tesla K40c[27],
- RAM: $256[GB]$ DDR4.

Los siguientes dos nodos, con nombres de host $Tokyo\_01$ y $Tokyo\_02$ contienen:

- CPU: 2x Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz,
- GPU: 2x Nvidia Tesla K40m[28],

[27] La tarjeta dispone de sistema de enfriamiento activo.

[28] La tarjeta dispone de sistema de enfriamiento pasivo.

- RAM: $256[GB]$ DDR4,

cada uno. Adicionalemente,

- El clúster cuenta con una capadidad de disco duro (mecánico) de $24[TB]$, aproximadamente.
- La conexión de red es ethernet con cables de red de $100[Mbps]$

4.1.1. Nvidia Tesla K40c/m Fabricada con la arquitectura Kepler [29], la tarjeta Tesla K40 dispone de las siguientes características:

- Cantidad de núcleos CUDA: 2880.
- Cantidad de SMs[30]: 15.
- Cantidad de núcleos CUDA por SM: 192.
- Frecuencia de los núcleos: $745[MHz]$.
- Memoria: $12[GiB]$[31] GDDR5.
- Frecuencia de memoria: $3003[MHz]$.
- Máximo ancho de banda de memoria: $288[GB/s]$.
- Máximo valor de desempeño: $4.29[TFLOPS]$.

---

[29] `https://www.nvidia.com/en-us/data-center/kepler-gpu-architecture/`. Accedido por última vez: 2018-11-28.

[30] Stream Multiprocessor. Agrupación de núcleos CUDA y sus respectivos recursos.

[31] Gibibytes.

4.1.2. Intel Xeon E5-2670 v3 De la cuarta generación de arquitecturas de Intel, Haswell, este procesador tiene las siguientes características:

- Cantidad de núcleos: 12C/24T[32].
- Frecuencia base: $2.30[GHz]$.
- Memoria: Hasta $768[GB]$ DDR4.
- Máximo valor de desempeño: $331.2[GFLOPS]$.

4.1.3. Intel Xeon E5-2620 v3

- Cantidad de núcleos: 6C/12T
- Frecuencia base: $2.40[GHz]$.
- Memoria: Hasta $768[GB]$ DDR4.
- Máximo valor de desempeño: $165[GFLOPS]$.

Las características del hardware a usar indican que las etapas de mayor costo computacional, en términos de operaciones de punto flotante, dentro del esquema de inversión se deben realizar dentro del procesador con el mayor valor de máximo de desempeño. Esto significa que las etapas de propagación y retropropagación, debido a que las operaciones necesarias para implementarlas implican una dependencia de datos (la salida de una función es la entrada de la siguiente), se debería destinar la implementación de estas etapas para funcionar sobre las GPUs.
En cuanto a la cantidad de memoria RAM por nodo, a cada GPU se le puede asignar poco menos de $128[GB]$ de RAM de host (si se ejecutan aplicaciones que usen

[32] C (cores): Físicos, T (threads): Lógicos. Nomenclatura definidad por Intel.

tanto memoria de GPU como de CPU). Por otra parte, la capacidad de disco duro no será muy relevante ya que, como se verá en las siguientes secciones, es conveniente que su uso sea el mínimo posible. La conexión de red disponible tampoco representa un problema debido a que, aún con dispositivos como InfiniBand o Intel Omnipath[33], el desempeño del método de inversión depende de qué metología se use para implementarlo.

4.2. Implementación de la propagación 3D usando CUDA-C

Ya se mencionó que las etapas de propagación y retropropagación son las de mayor costo computacional al interior del método de inversión. Adicionalemente, estas etapas se pueden generalizar como un conjunto de operaciones sobre arreglos de datos y gracias a esto es posible utilizar kernels de CUDA para representarlos. La función de propagación se compone de varios kernels, sin embargo, los de mayor importancia son[34] (ver figura 17):

- `kernelP();`: Kernel usado para calcular el campo P, cada muestra de tiempo, a partir de las derivadas en las direcciones x, y y z de de los campos V_x, V_y y V_z, respectivamente.

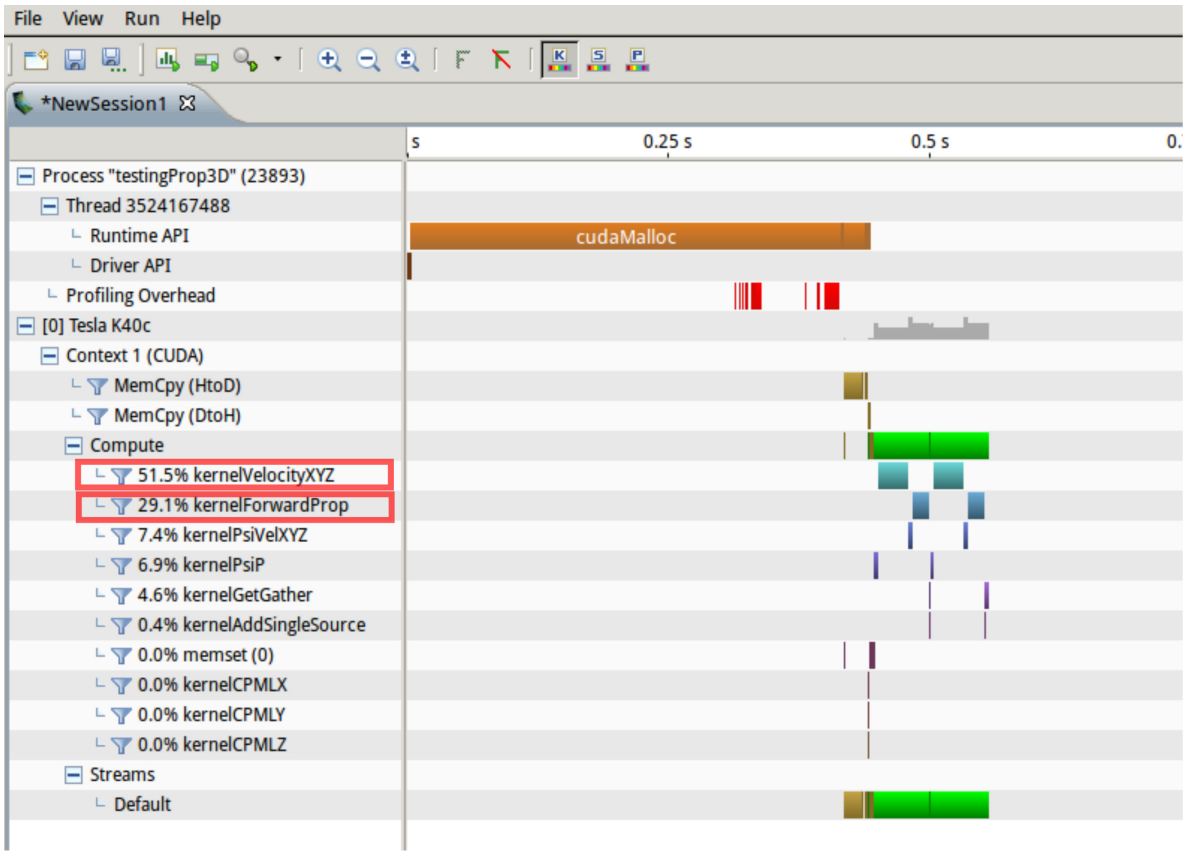
- `kernelVelXYZ();`: Kernel usado para calcular , en cada muestra de tiempo, los campos V_x, V_y y V_z a partir de las derivadas en las direcciones x, y y z, respectivamente, del campo$P$.

---

[33] `https://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html`. Accedido por última vez: 2018-11-28.

[34] NVIDIA ofrece una herramienta llamada Nividia Visual Profiler en la instalación del entorno de CUDA. Esta herramienta clasifica los kernels de cualquier implementación en términos del tiempo de ejecución que representan respecto al tiempo total de ejecución de la aplicación.

Figura 17. Imagen ejemplo de la clasificación que realiza Nvidia Visual Profiler de los kernels ejecutados en la aplicación. El porcentaje de tiempo frente al nombre de cada kernel (encerrados en recuadros rojos) es relativo al tiempo total de la aplicación implementada. Esta clasificación está incluída dentro del análisis que ejecuta la herramienta para indicar cuáles son los puntos críticos de la aplicación.



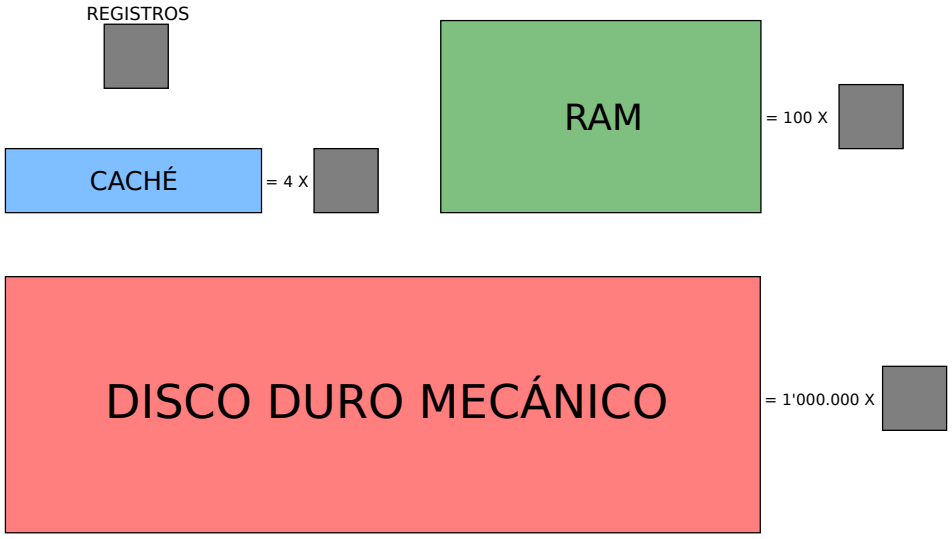
También se han creado otros kernels para sumar la fuente en cada muestra de tiempo, calcular la información de las fronteras no naturales (CPML), extraer el dato modelado, etc, pero el porcentaje de relevancia es muy bajo en comparación con los listados previamente.

En un escenario de procesamiento de datos de talla real[35] se requieren terabytes

[35] Modelos de velocidades de dimensiones $500 \times 500 \times 140$ o superior

de capacidad de memoria para generar los gradientes con los que se actualizan los modelos de densidad y velocidad. Sin embargo, más allá de tener un alto espacio para datos, se debe revisar la penalización de tiempo en cada uno de los niveles de la jerarquía de memoria para decidir cómo se va a enfocar la implementación. En la figura 18 se muestran las proporciones del tiempo que se necesita para acceder a los distintos niveles de memoria respecto al tiempo de acceso a registros (el tipo de memoria con tiempo de acceso, para lectura o escritura, de hasta un ciclo de reloj del procesador).

Figura 18. Comparación de los tiempos de acceso a los diferentes niveles de memoria. Nótese que los accesos a disco duro son excesivamente costosos y se debe tener un uso limitado de este independientemente de la aplicación. Imagen creada a partir de [36] y [37].



Teniendo esto en cuenta, se plantea una primera prueba con la configuración de parámetros mostrados en la tabla 1. Se tomó como ejemplo las dimensiones del modelo procesado en el trabajo de [38].

---

[38] David L Abreo, Sergio A Abreo y Ana B Ramirez. “A hybrid methodology for a 3D Full Waveform Inversion in time domain using GPUs”. En: *15th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 31 July-3 August 2017*. Brazilian Geophysical

Tabla 1. Parámetros de inicialización para el primer experimento del propagador.

Variable	Descripción	Valor
N_x	Puntos del modelo en la dimensión x	250
N_y	Puntos del modelo en la dimensión y	250
N_z	Puntos del modelo en la dimensión z	140
N_t	Muestras de tiempo	2000

Sobre un modelo uniforme con velocidad de $1500[m/s]$ y densidad de $1010[kg/m^3]$ (aunque estos parámetros son independientes de las métricas a generar) se midió el tiempo de ejecución para diferentes experimentos tal como se muestra en la tabla 2.

Tabla 2. Comparación de tiempo de ejecución de la primera versión de la función de propagación. La penalización de tiempo es relativa al experimento 1.

Experimento	Tiempo de ejecución [s]	Penalización de tiempo [$\%$]	Bytes escritos
1	67.80	no aplica	no aplica
2	74.04	9.2	23[MB]
3	385.02	467.87	280[GB]

- Experimento 1: Solo se realizó el cálculo del dato modelado sin realizar transferencias entre RAM de host y device. El dato modelado se obtiene guardando la información de la cara superior del cubo que representa cada muestra de tiempo dentro de la zona de interés (no se guarda información dentro de las zonas de CPML). No se guardaron datos en disco de la información de los hipercubos.

- Experimento 2: Se realizó el cálculo del dato modelado realizando transferencias entre host y device cada muestra de tiempo. Adicionalmente, se almacenó en disco duro el dato modelado.

Society. 2017, págs. 369-373.

- Experimento 3: Se almacenó en disco la información de los cuatro hipercubos $\mathbf{P}$, $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$ mientras se generaba el dato modelado. Adicionalmente, se almacenó en disco duro el dato modelado.

Nótese que el incremento del tiempo de ejecución asociado a transferir el dato modelado entre memorias de host y device y almacenar esta información en disco duro, es insignificante respecto al tiempo desperdiciado por guardar la información de los hipercubos de propagación en disco duro. Es necesario aclarar que durante esta prueba no habían otros procesos escribiendo datos en disco duro, de lo contrario el tiempo de ejecución sería aún mayor para el experimento tres. Para recrear un escenario de modelos de talla real,se estima que la inversión de un modelo de dimensiones $500 \times 500 \times 140$, para un campo de propagación de 5000 muestras de tiempo, usando 200 fuentes y realizando 10 iteraciones del método, podría tardar alrededor de un año y diez meses si se almacena la información tal como se hace en el experimento tres.

Con este primer resultado se sugiere evitar (por mala práctica de implementación) hacer uso constante del espacio en disco (principalmente si es un disco duro mecánico). Aunque el tiempo de acceso a RAM sigue siendo alto en comparación con el de los registros, su capacidad de almacenamiento es aceptable y se ubica como un punto intermedio entre estas dos características. Sin embargo, el sistema de trabajo no dispone de más de $5[TB]$ de RAM en ninguno de los nodos de trabajo (tampoco si se calcula el total de RAM del cluster), por lo tanto la estrategia a definir debe incluir una representación en forma comprimida de los hipercubos de datos que se generan en las etapas de propagación.

La implementación de esta primera versión tiene en cuenta aspectos básicos de programación en paralelo, usando CUDA, como el acceso a elementos consecutivos en memoria por parte de threads de índice consecutivo (ilustrado en la figura 19).

4.2.1. Estabilidad y dispersión numérica Al discretizar ecuaciones diferenciales mediante diferencias finitas se deben tener en cuenta efectos como la estabilidad y dispersión numérica. Para la versión de la función de propagación implementada, la estabilidad se garantiza teniendo en cuenta la expresión 21 según [39]

$$\frac{\Delta t \cdot C_{max}}{\Delta h} \leq \frac{1}{\sqrt{3}\sum_{l=1}^{L} |d_l|} \tag{21}$$

donde Δt es el paso temporal, Δh es el menor valor entre Δx, Δy y Δz (los pasos espaciales usados en la simulación), d es el vector que contiene los coeficientes de diferencias finitas de octavo orden y C_{max} es la velocidad máxima dentro del modelo a procesar.

Por otro lado, la dispersión numérica generalmente es influenciada por el ángulo de propagación, el orden de aproximación espacial y la cantidad de puntos por longitud de onda. El ángulo de propagación es difícil de controlar y el esquema de aproximación espacial está fijo en octavo orden. Debido a esto, la dispersión numérica se puede controlar asegurando que se tiene un muestreo espacial denso en el campo de propagación. Para esto, según [40], se deben garantizar por lo menos cinco puntos por longitud de onda ($pplo$) teniendo en cuenta la expresión 22

$$\Delta h = \frac{C_{min}}{pplo \cdot 2 \cdot f_{max}} \tag{22}$$

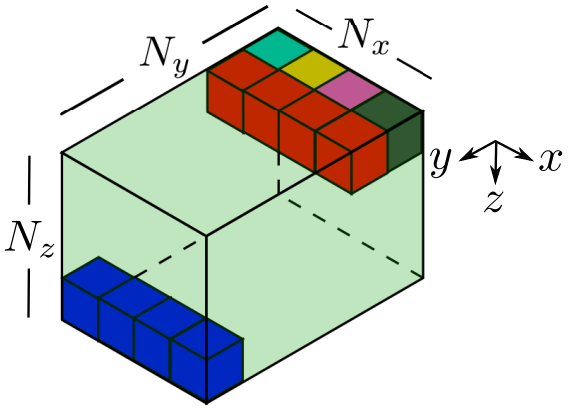
donde C_{min} es el valor mínimo del modelo de velocidades y f_{max} es la frecuencia máxima de la señal usada como fuente.

---

[39] XM Wang. “Seismic wave simulation in anisotropic media with heterogeneity using a high-order finite-difference method”. En: (2001).

[40] Herling Gonzales. “Taller de Modelado de Propagación de Ondas en Aproximación Escalar y Vectorial Basado en Malla Intercalda (Staggered Grid)”. 2016.

Figura 19. Acceso, para lectura o escritura, a los datos en la RAM tanto de device como de host. La variable de acceso más rapida es x, seguido de y y de z en ese orden. idx es la variable que contiene el índice de acceso al cubo de datos en RAM.



$$idx = x + N_x \times y + N_x \times N_y \times z$$

Datos en RAM

4.3. Estrategias de manejo de memoria

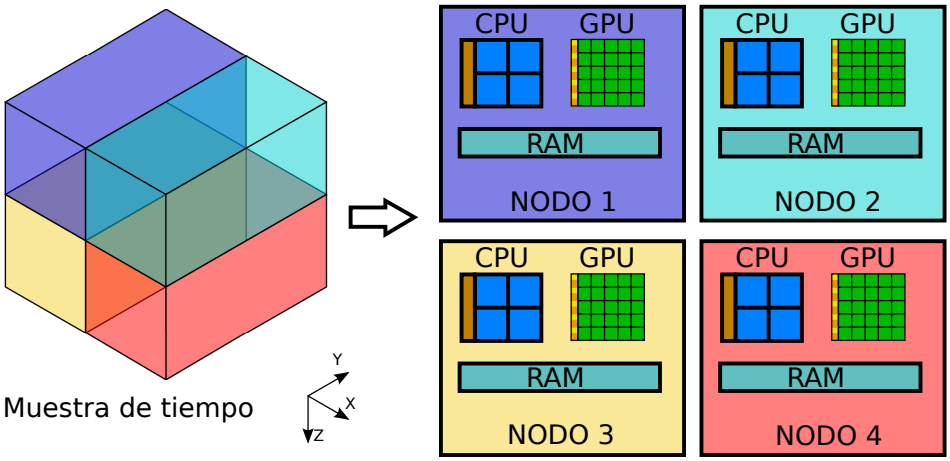
En el estado-del-arte existen estrategias para lidiar con el problema de la cantidad de memoria consumida por la etapa de propagación. Estas estrategias tienen en cuenta tanto el hardware disponible para realizar la simulación como la forma de expresar los datos al interior de la implementación.

La descomposición de dominio [41] es una estrategia computacional que permite distribuir la simulación del modelado de la ecuación de onda sobre múltiples nodos de cómputo (ver figura 20). Cada uno de estos nodos calcula una porción de cada muestra de tiempo (cubo de datos), permitiendo modelar la interacción del campo

[41] Huan-Ting Meng y Jian-Ming Jin. "Acceleration of the Dual-Field Domain Decomposition Algorithm Using MPI–CUDA on Large-Scale Computing Systems". En: *IEEE Transactions on Antennas and Propagation* 62.9 (2014), págs. 4706-4715.

de presión de modelos muchos más grandes de los que se podría con un solo nodo. Sin embargo, cómo se acaba de mencionar, la limitación es la cantidad de nodos que dispone el sistema de cómputo.

Figura 20. Ejemplo de aplicar descomposición de dominio para calcular una muestra del hipercubo de datos del campo de propagación. Para el caso de cuatro nodos de cómputo, el volumen que representa la muestra se divide en cuatro secciones y cada sección es calculada en un nodo diferente.



Por otro lado, la estrategia de reconstrucción de campo (aprovechada en la implementación de la FWI (usando GPUs) en el dominio $2D$ [42] y $3D$ [38]) hace posible reducir el consumo de memoria de las etapas de propagación y retropropagación a menos del $5\,\%$ del valor inicial pero con una penalización del tiempo de ejecución del $55\,\%$[43] debido a que se necesita una etapa adicional de propagación para generar el gradiente.

La figura 21 ilustra cómo funciona la estrategia de reconstrucción, la cuál se resume

---

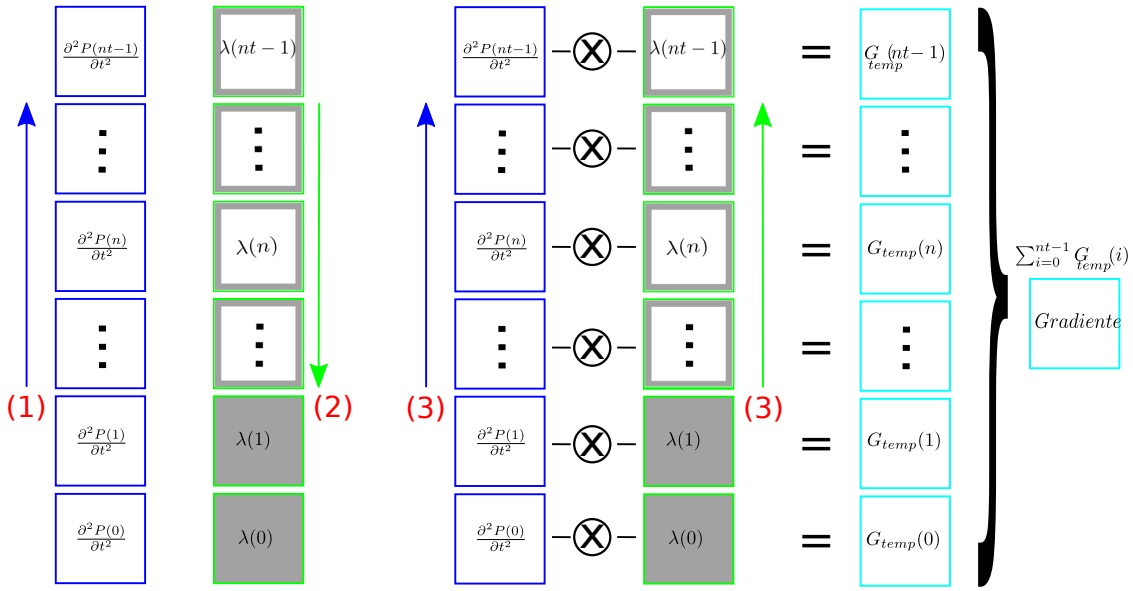
[42] Reynaldo F. Noriega y col. “Implementation strategies of the seismic Full Waveform Inversion”. En: *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE. 2017, págs. 1567-1571.

[43] Es necesario aclarar que en los trabajos realizados no se han considerado las transferencias entre RAM del host y RAM del device y los accesos a disco duro. Los datos fueron procesados en la RAM de la GPU.

a continuación:

- Primera etapa: Realizar la propagación para obtener el dato modelado con el cuál se calculan los datos residuales.
- Segunda etapa: Mientras se realiza la retropropagación de los residuales, se almacena en la RAM de la GPU la información asociada a los bordes de la zona de interés dentro del modelo en cada muestra de tiempo del campo $\mathbf{P}$.
- Tercera etapa: Con un paso de simulación adicional, se reconstruye la información de la retropropagación mientras se calcula nuevamente la propagación y se calcula el gradiente necesario para actualizar el modelo.

Figura 21. Estrategia de reconstrucción de campo aplicada en las implementaciones de FWI 2D y 3D mostrados en [44] y [45], respectivamente. La zona en color gris representa la información que se va a guardar de cada muestra de tiempo del campo de retropropagación. Las flechas de color azul y verde indican cómo se calcula cada campo y los números en rojo indican el orden en que se hace cada simulación (primero propagación, luego retropropagación y finalmente se reconstruye la retropropagación mientras se recalcula la propagación).



No se reconstruye el campo de propagación debido a que la derivada en el tiempo amplifica los errores numéricos que se generan por la estrategia y se pueden proyectar como artefactos en el modelo final después del proceso de inversión.
La cantidad de memoria usada por el algoritmo de inversión, para guardar los hipercubos asociados a la propagación y retropropagación, está dada por la expresión (23)

$$RAM_{old}[GB] = 2 \times \frac{N_x \times N_y \times N_z \times N_t}{1 \times 10^9} \tag{23}$$

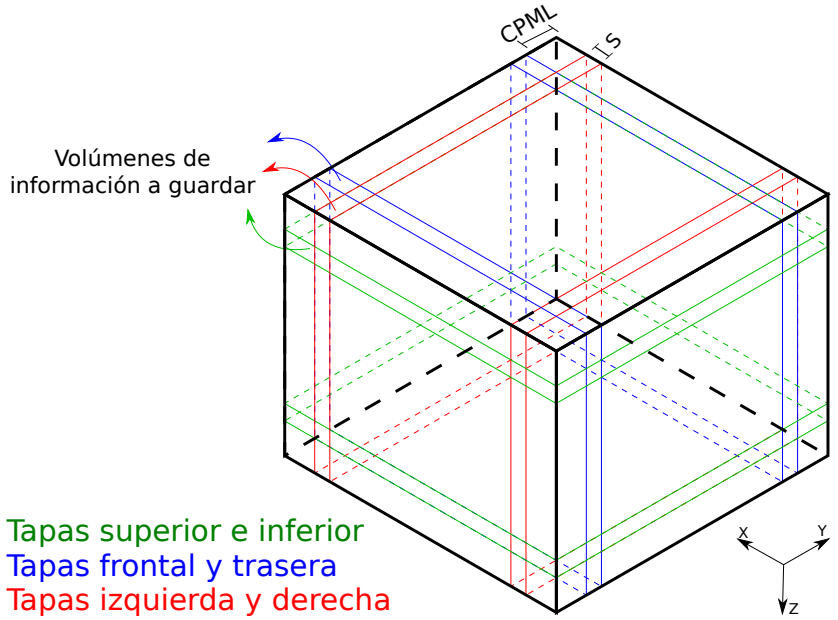
donde N_x, N_y y N_z son las tres dimensiones de los modelos de velocidad y densidad y N_t es la cantidad de muestras de tiempo.
Cuando se implementa la estrategia de reconstrucción, la cantidad de memoria que se utiliza está dada por la expresión (24)

$$RAM_{new}[GB] = \frac{2 \times \mathrm{s} \times (N_x \times N_y + N_x \times N_z + N_y \times N_z) \times N_t}{1 \times 10^9}, \tag{24}$$

donde s es la mitad del orden de la aproximación de diferencias finitas utilizada (8 para este caso, entonces $\mathrm{s} = 4$), $2\mathrm{s} \times N_x \times N_y$ son las dimensiones de las *tapas* superior e inferior de cada muestra del hipercubo, $2\mathrm{s} \times N_x \times N_z$ son las dimensiones de las *tapas* frontal y trasera de cada muestra del hipercubo y $2\mathrm{s} \times N_y \times N_z$ son las dimensiones de las *tapas* izquierda y derecha de cada muestra del hipercubo (ver figura 22).
Asumiendo el escenario $N_x = N_y = N_z$, la relación entre la memoria usada por el método de inversión cuando se usa la estrategia de reconstrucción y cuando la implementación almacena los hipercubos de datos de la propagación está data por la expresión (25)

$$\frac{RAM_{new}}{RAM_{old}} = \frac{3\mathrm{s}}{N_x} \tag{25}$$

Figura 22. Una muestra t_m del hipercubo de datos a reconstruir. La información dentro de los paralelepípedos de colores es la que se guarda en RAM para la posterior reconstrucción del campo.



También existe la estrategia computacional, conocida como check-pointing [46], usada para reducir el uso de memoria de la etapa de propagación. Sin embargo, esta estrategia incurre en una penalización del tiempo de ejecución de hasta $1000\,\%$ si no se tiene cuidado con el porcentaje de reducción de memoria utilizada que se desea. Otros trabajos en el estado-del-arte presentan estrategias como utilizar el dominio de la frecuencia para procesar datos de talla real [47] (sin tener en cuenta el costo computacional) o mezclar el dominio de la frecuencia y del tiempo para reducir el

---

[46] David Imbert y col. “Tips and tricks for Finite difference and i/o-less FWI”. En: *SEG Technical Program Expanded Abstracts 2011*. Society of Exploration Geophysicists, 2011, págs. 3174-3178.

[47] Patrick Amestoy y col. “Efficient 3D frequency-domain full-waveform inversion of ocean-bottom cable data with sparse block low-rank direct solver: a real data case study from the North Sea”. En: *SEG Technical Program Expanded Abstracts 2015*. Society of Exploration Geophysicists, 2015, págs. 1303-1308.

consumo de memoria [48] (para procesar conjuntos de datos muy pequeños en comparación con las dimensiones que implican los datos reales) del método de inversión 3D.

Teniendo identificados los trabajos en el estado-del-arte, aplicar la metodología de reconstrucción de campo se sitúa como lo más conveniente para el desarrollo de la estrategia computacional que se propone en este trabajo de investigación. El método de reconstrucción de campo se escoge tanto por el hardware del que se dispone como por su beneficio en términos del ahorro de memoria dentro del esquema inversión. Aunque la estrategia de reconstrucción se presenta aplicada sobre la ecuación de onda acústica con densidad constante, su uso se puede extrapolar para reconstruir la información de propagación de la ecuación de onda con densidad variable.

[48] Lu Liu y col. "3D hybrid-domain full waveform inversion on GPU". En: *Computers & Geosciences* 83 (2015), págs. 27-36.

5. ESTRATEGIA COMPUTACIONAL DE IMPLEMENTACIÓN DE LA FWI 3D

5.1. Definición de la estrategia de implementación

Hasta este punto se ha mostrado cómo funciona el método de inversión de manera general y su elevado costo computacional, los APIs y los entornos utilizados en el área de HPC y las técnicas en el estado del arte más relevantes para el manejo de memoria dentro de un sistema de cómputo.

Para cumplir el objetivo de este trabajo se dispone de un clúster de cómputo heterogéneo[49] y se tiene un método de manejo de memoria como punto de inicio (estrategia de reconstrucción de campo). Adicionalmente, es necesario que la implementación a desarrollar aproveche los beneficios de la capacidad de almacenamiento a la que puede acceder la CPU y el paralelismo que ofrecen las GPUs.

Por lo tanto, la estrategia computacional de implementación de la FWI 3D propuesta en este trabajo está enmarcada por las siguientes características:

- Manejo de almacenamiento de la información:
  - Se va a administrar el consumo de memoria a nivel de un nodo de cómputo. Debido a esto se usará la estrategia de reconstrucción de campo para disminuir la RAM utilizada por el método de inversión.
  - Los volúmenes necesarios para las etapas de propagación, retropropagación y cálculo del gradiente (según las ecuaciones (1) a (4) y las expresiones (14) a (17)) serán reservados en la RAM de la GPU.

---

[49] Se considera clúster heterogéneo o equipo de computación heterogénea a aquel sistema con más de un tipo de procesador (`https://www.linuxjournal.com/article/8368`. Accedido por última vez: 2018-11-28). Para este caso se disponen de CPUs y GPUs.

- Volúmenes asociados a la información de fronteras para la estrategia de reconstrucción, el método L-BFGS, modelos de velocidad y densidad serán reservados en RAM de host.
- Debido a que los datos observados pueden abarcar centenares de gigabytes, es necesario que esta información se lea desde disco duro cada iteración de la FWI 3D pues los accesos a RAM principal se consideran más convenientes para otras variables de mayor frecuencia de uso.

- Complejidad computacional:
  - Las etapas de propagación, retropropagación y cálculo del gradiente se realizan dentro de la GPU debido a que son los procesos de mayor complejidad computacional ($O(N^4)$) dentro del esquema de inversión .
  - Procesos como cálculo del dato residual, método L-BFGS y actualización de los modelos de velocidad y densidad se realizan a nivel de CPU debido a su menor costo computacional (complejidad computacional de hasta $O(N^3)$). Se hará uso del API de OpenMP para acelerar estos procesos durante la ejecución del método de inversión.
  - El impacto al tiempo de ejecución, producto de la estrategia de reducción de consumo de memoria, se maneja con el uso del API de MPI. Al disponer de un sistema de cómputo con más de un nodo, se usará MPI para calcular tantos gradientes como GPUs hayan disponibles (para un máximo de seis con el hardware actual).
  - Se usa el método L-BFGS para ayudar en la convergencia de la técnica de inversión.

- Transferencias de datos:
  - Durante las etapas de propagación y retropropagación se harán transfe-

rencias de datos que representan alrededor del $1\,\%$ de los datos que se procesan en éstas etapas.

- A nivel de cluster, se va a usar el API de MPI para acumulación de variables generadas en cada nodo (e.g. Acumular los gradientes independientes obtenidos en cada GPU como un solo gradiente, el cuál se usa para actualizar los modelos de velocidad y densidad o para el método de L-BFGS).

5.2. Incremento de rendimiento de la función de propagación

Aunque el objetivo principal de este trabajo es el manejo de memoria dentro de un clúster que utiliza GPUs, en la estrategia se debe tener en cuenta cómo se penaliza el tiempo de ejecución[50]. Es por esto que se propone mejorar el rendimiento de la etapa de propagación (como es la etapa que implicará el tiempo de ejecución más alto, reducir este valor[51] beneficiará el proceso en general) y aplicar estas mejoras en la posterior etapa de retropropagación[52] para reducir el impacto al tiempo de ejecución.

Como ya se mencionó, la propagación se desarrollará mediante $kernels$ de CUDA y para aprovechar el alto rendimiento de las GPUs es necesario revisar en el interior de estos dispositivos y encontrar características útiles para la implementación. La figura 23 muestra la arquitectura general de una GPU de NVIDIA. Se puede apreciar

---

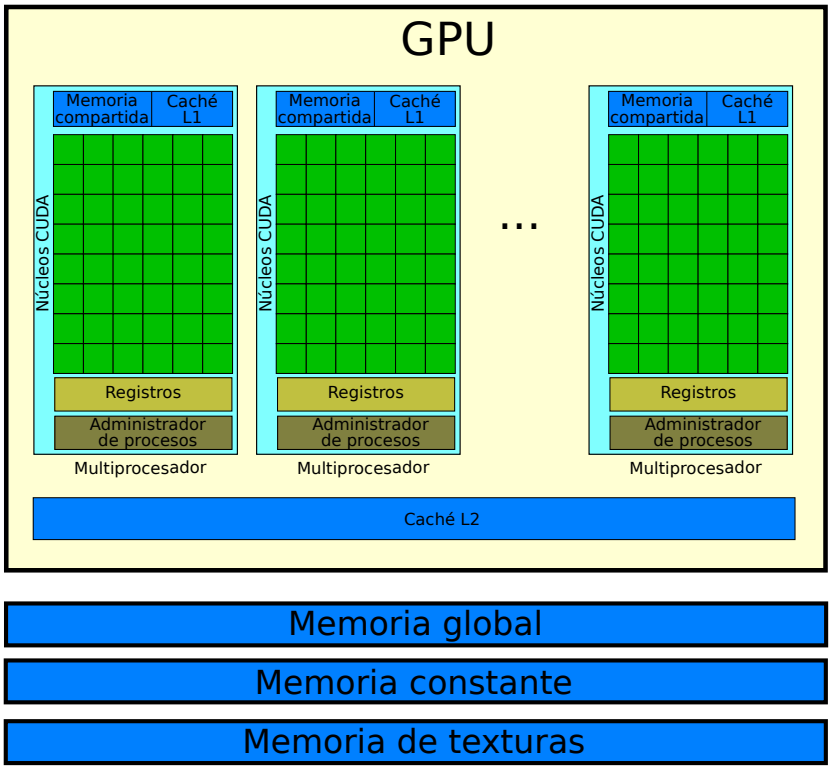
[50] Un ejemplo de estrategia podría ser almacenar constantemente toda la información en disco duro pero ya se vió que esto significaría una pérdida de tiempo y de recursos por mal manejo de los mismos.

[51] Con la ayuda de las herramientas de análisis de NVIDIA se pueden revisar los puntos críticos en la ejecución de cada $kernel$ para su posterior optimización. No existe una receta específica para este proceso porque depende del objetivo del $kernel$ implementado.

[52] El operador es autoadjunto y permite utilizar los mismos $kernels$ en CUDA-C para el cálculo del campo de retropropagación.

que además de la RAM (memoria global en términos del entorno de CUDA), existen otros tipos de memoria como la memoria compartida, la memoria de texturas y la memoria constante. Si se aprovechan las ventajas que ofrecen cada uno de estos tipos de memoria, se incrementará el rendimiento de los $kernels$ implementados.

Figura 23. Esquema general de la arquitectura de una GPU. En el diagrama se pueden apreciar la unidad más básica conocida como núcleo CUDA y los diferentes tipos de memoria dentro del dispositivo.



5.2.1. Distribución y cantidad de $threads$ ejecutándose La unidad básica de ejecución en CUDA se llama thread, un grupo de $threads$ conforman un bloque y un grupo de bloques forman un $grid$ de miles de tareas ejecutándose en parale-

lo. Cualquier GPU con arquitectura Fermi [53] o superior soporta bloques de máximo $1024\ threads$ y se pueden crear bloques 3D con un valor máximo de 1024, 1024 y 64 $threads$ en las dimensiones x, y y z, respectivamente. Para obtener el mejor desempeño, controlando sólo el número de $threads$, se recomienda usar una distribución tal que la cantidad de $threads$ por bloque esté entre 128 y 256[54].

Por otro lado, surge la pregunta de cómo organizar estos $threads$ (e.g. Â¿Es más conveniente usar un grid[55] 3D para acceder a los datos en lugar de un $grid$ 1D?. Ver figura 24). En tiempo de ejecución, el hardware tiene que administrar el proceso de destrucción y creación de $threads$ para terminar el $kernel$ implementado. Es decir, cada $thread$ tiene un contexto de trabajo y por ende un costo en tiempo. Es por ello que esta parte del trabajo tuvo un enfoque de prueba y error para encontrar la mejor acomodación de $threads$ que se lanzaban por cada $kernel$ implementado.

5.2.2. Transacciones de memoria En CUDA existen los $warps$, que consiten en grupos de $32\ threads$ que deben realizar siempre el mismo proceso en la ejecución de un kernel. Una transacción de memoria en una GPU es el acceso, mediante la misma operación de memoria, de un $warp$ a un bloque consecutivo de 32 elementos (uno por cada thread). Si esto no se cumple (ver figura 25), se va a segmentar el proceso en más de una operación incrementando el tiempo de ejecución.

Teniendo esto en cuenta, la dimensión más rápida de acceso al cubo que se está procesando (generalmente la dimensión x) debe ser un múltiplo del tamaño de un

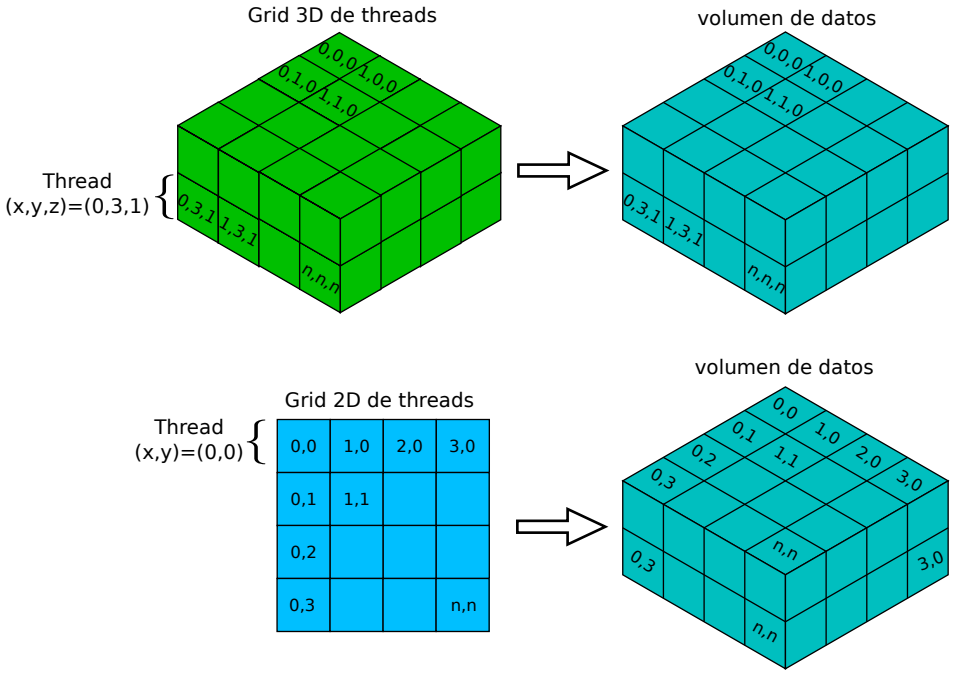
---

[53] `https://www.nvidia.com/content/PDF/fermi_white_papers/`. Accedido por última vez: 2018-11-28.

[54] Michael Vollmer y col. “Meta-programming and Auto-tuning in the Search for High Performance GPU Code”. En: *Proceedings of the 4th ACM SIGPLAN Workshop on Functional High-Performance Computing*. ACM. 2015, págs. 1-11.

[55] Malla o rejilla de elementos

Figura 24. Comparación del uso de un $grid$ 2D y un $grid$ 3D de $threads$ para acceder a un volumen de datos. En el $grid$ 2D el mismo $thread$ accede a dos posiciones de memoria mientras que en el grid 3D cada $thread$ accede a una sola posición. También es posible crear un $grid$ 2D que contiene la misma cantidad de $threads$ que el $grid$ 3D pero accediendo a los datos con un índice diferente.



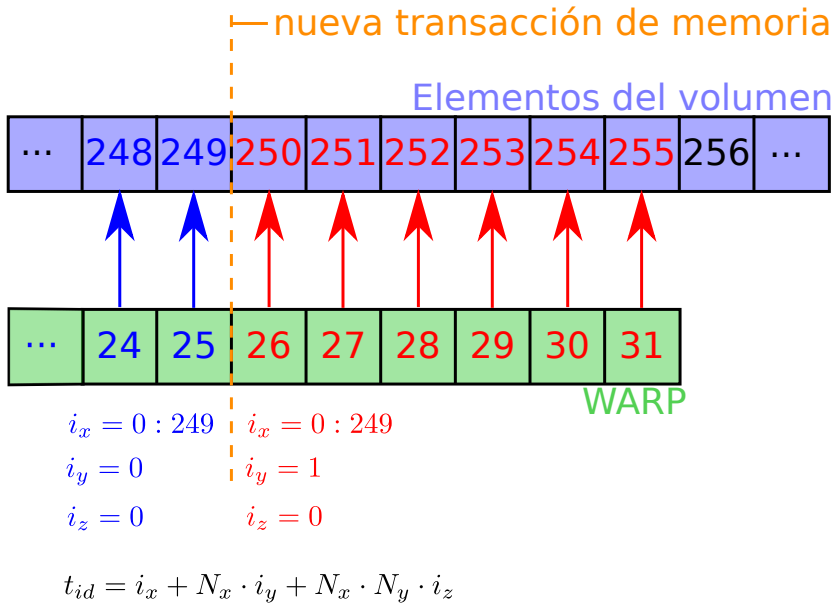
$warp$ (múltiplos de 32). De esta forma, los parámetros de la tabla 1 se modifican tal como se ve en la tabla 3.

Tabla 3. Parametros de inicialización ajustados para tener el menor número de transacciones de memoria en los $kernels$ de CUDA.

Variable	Descripción	Valor
N_x	Puntos del modelo en la dimensión x	256
N_y	Puntos del modelo en la dimensión y	250
N_z	Puntos del modelo en la dimensión z	140
N_t	Muestras de tiempo	5000

5.2.3. Memoria constante Este tipo de memoria se encuentra fuera del chip de la GPU, es de sólo lectura y tiene su propia caché. La ventaja de usar este

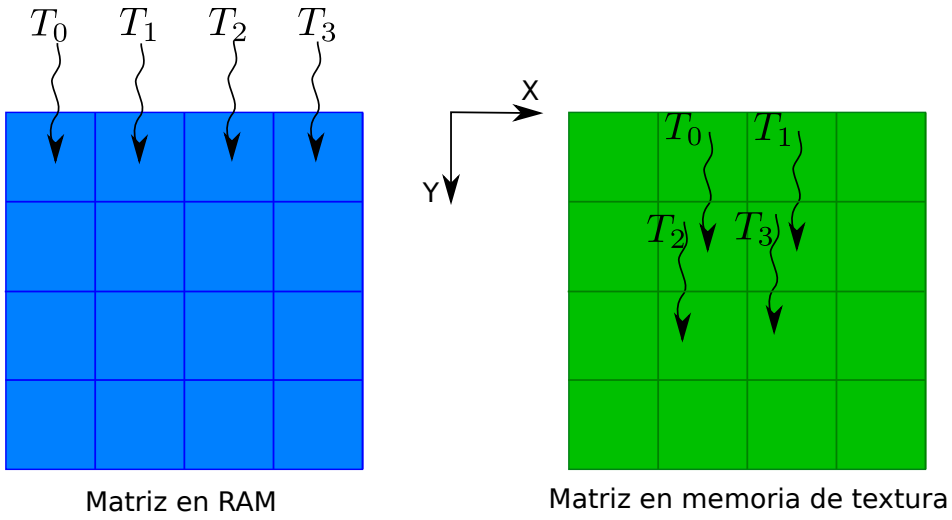
Figura 25. Ejemplo de problema de transacciones de memoria. Los últimos seis $threads$ de un determinado $warp$ van a realizar la operación de acceso a memoria con un valor de i_y diferente a los primeros 26 $threads$ dentro del mismo warp. Esta diferencia de acceso dentro de un mismo $warp$ produce que el dispositivo invierta más ciclos de reloj leyendo o escribiendo datos en RAM.



espacio es que se reduce la cantidad de registros usados por cada thread. Para el caso específico de la función de propagación, los coeficientes de la aproximación de diferencias finitas pueden ser almacenados en la memoria constante antes de lanzar el kernel. De esta forma, no se invierten recursos en variables que contengan estos valores sino que se acceden para lectura, lo cuál deja más registros libres por $thread$ ejecutado, incrementando el rendimiento del $kernel$ implementado.

5.2.4. Memoria de texturas Aunque es de sólo lectura, la memoria de texturas tiene un amplio uso en el desarrollo de videojuegos, debido a su diseño para procesamiento de imágenes (Matrices o arreglos 2D). Su característica principal es el desempeño en términos de localidad espacial al momento de acceder a los datos (ver figura 26).

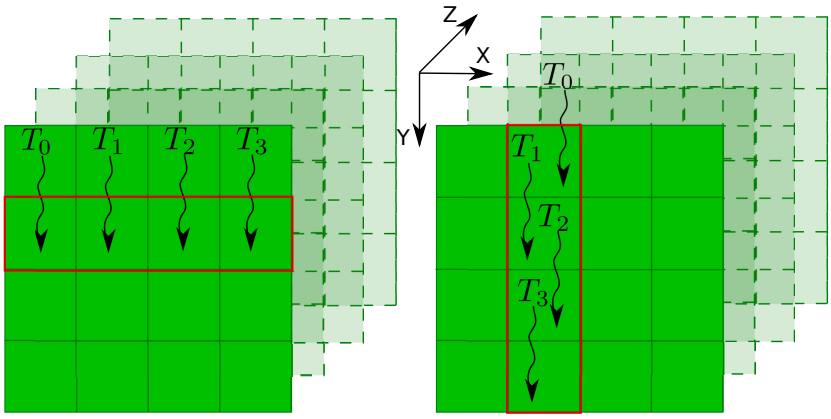
Figura 26. Ilustración de cómo cuatro $threads$ (flechas distorsionadas) acceden a una matriz (ordenada por filas) almacenada en RAM (azul) y a la memoria de texturas (verde). En la memoria de texturas se aprovecha la localidad espacial en ambas direcciones y los $threads$ dentro de un mismo $warp$ no tendrán problemas de acceso a memoria.



Adicionalmente, aprovecha el canal de comunicación entre la caché y la GPU. Teniendo esto en cuenta, para los $kernels$ que aplican las derivadas espaciales en las direcciones x y y se utiliza la memoria de texturas para acceder a los arreglos de entrada (ver figura 27). Sin embargo, no se puede abusar de esta característica y aplicarla a cualquier $kernel$ implementado. Para el caso de la función que realiza el cálculo de la derivada espacial en la dirección z, el uso de memoria de texturas disminuirá el rendimiento del $kernel$ debido a que los datos que se acceden para calcular cada punto de la salida no están organizados como una matriz o arreglo 2D (caso contrario con los $kernels$ que calculan las derivadas en las direcciones x y y).

5.2.5. Memoria compartida Con las mismas características de la cache (poco tiempo de acceso y capacidad de almacenamiento), la memoria compartida tiene la ventaja de que puede ser accedida para escritura por instrucciones desarrolladas en CUDA-C. De esta forma, datos que son accedidos múltiples veces desde la RAM de

Figura 27. Ventaja de usar la memoria de texturas para acceder los datos de entrada en los $kernels$ de las derivadas en las direcciones x y y. Los elementos en esas dimensiones son los que se almacenan primero que los de acceso en la dimensión z.



la GPU (ver figura 28) pueden ser copiados a la memoria compartida, lo que permite incrementar el rendimiento del $kernel$ implementado.

5.3. Estrategia de reconstrucción de campo: Ecuación de onda acústica con densidad variable

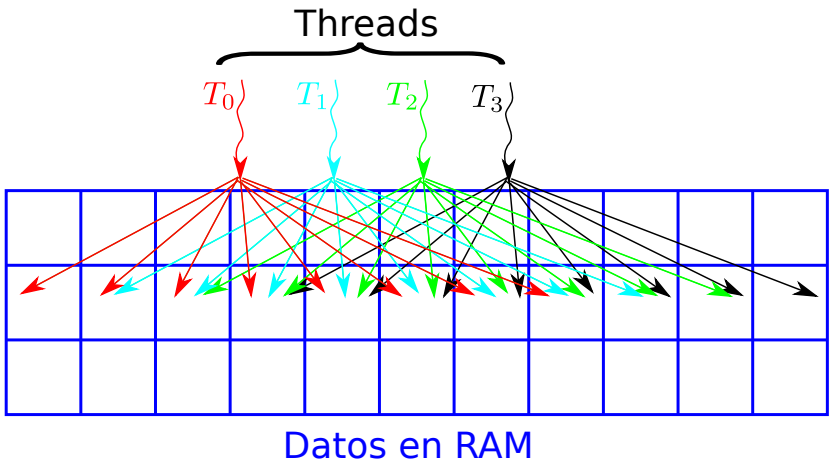
En el capítulo anterior se mostró cómo funciona la estrategia de reconstrucción según los trabajos de ?? y ?? (usando la ecuación de onda acústica en los dominios 2D y 3D, respectivamente). Este mismo método puede ser aplicado para usarse con el operador onda acústica con densidad variable mostrado en este trabajo (ver figura 29).

La expresión (26) indica la cantidad de RAM a utilizar por el método de inversión sin tener en cuenta la reconstrucción.

$$RAM_{old}[GB] = 8 \times \frac{N_x \times N_y \times N_z \times N_t}{1 \times 10^9} \tag{26}$$

Cuando se aplica la estrategia de reconstrucción, la RAM usada es calculada con la

Figura 28. Ejemplo de redundancia de acceso a datos almacenados en RAM de la GPU. Cuatro $threads$ necesitan acceder a ocho elementos, cada uno, en RAM para calcular un elemento de la salida. La redundancia se refleja en la lectura, por más de un $threads$, al mismo elemento. Se podría ralizar una única copia de datos desde la RAM a la memoria compartida de la GPU y aunque se siga teniendo redundancia de acceso a datos, el tiempo de lectura de la memoria compartida (equivalente a la chaché) es mucho menor que el tiempo necesario para leer un dato almacenado en RAM.



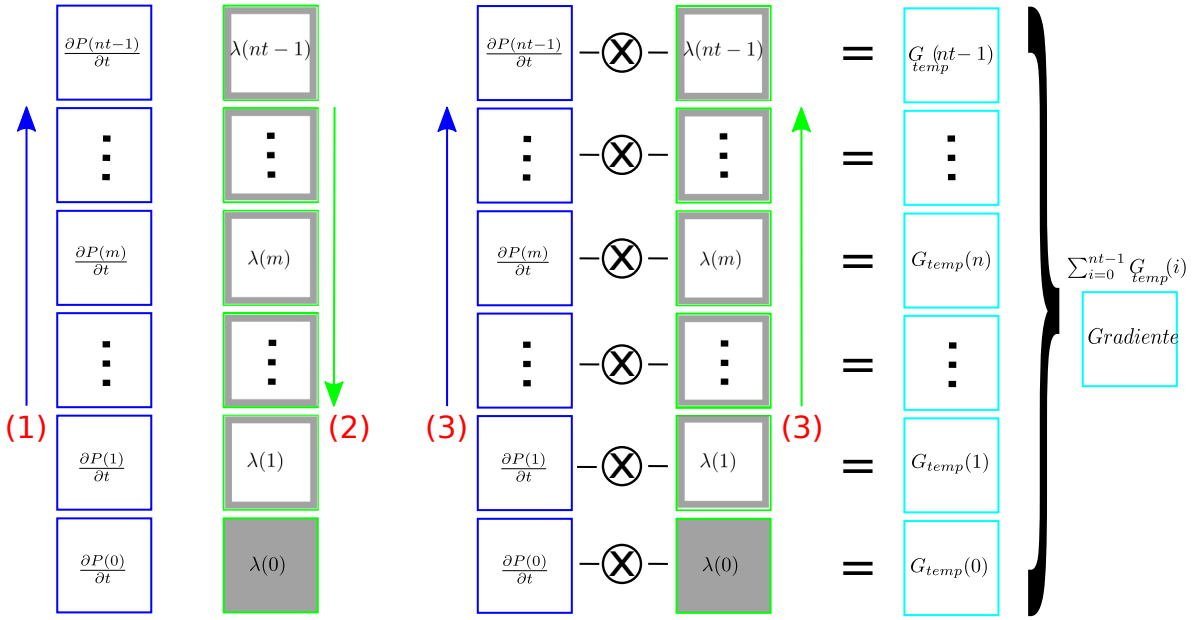
ecuación (24) y la relación entre las ecuaciones (26) y (24) (asumiendo el escenario $N_x = N_y = N_z$) está dada por la expresión (27).

$$\frac{RAM_{new}}{RAM_{old}} = \frac{3}{4}\frac{\mathbf{s}}{N_x}, \tag{27}$$

Nótese que para la ecuación de onda usada en este caso, el ahorro de memoria será mayor, lo cuál es una ventaja para la implementación.

5.3.1. Mejora de la estrategia de reconstrucción de campo: Propiedad de simetría del producto interno El producto interno existe como una generalización del producto punto pero aplicado sobre espacios vetoriales. Incluye las mismas propiedades del producto interno como multiplicación por un factor constante, propiedad conmutativa, etc. En notación de producto interno, las ecuaciones del cálculo de los gradientes de velocidad y densidad (18 y 19) se pueden reescribir como se aprecia

Figura 29. Proceso de reconstrucción de campo cuando se utiliza la ecuación de onda acústica con densidad variable. Como se necesita la información de la derivada respecto al tiempo del campo de propagación es más conveniente reconstruir el campo de retropropagación para evitar problemas de error numérico.



en las expresiones 28 y 29

$$G_{\mathbf{c}} = \frac{-2}{\rho \mathbf{c}^3} < \frac{\partial \mathbf{P}}{\partial t}, \lambda > \tag{28}$$

$$G_{\rho} = \frac{-1}{\rho^2 \mathbf{c}^2} < \frac{\partial \mathbf{P}}{\partial t}, \lambda > + < \frac{\partial \mathbf{V_x}}{\delta t}, \lambda_x > + < \frac{\partial \mathbf{V_y}}{\delta t}, \lambda_y > + < \frac{\partial \mathbf{V_z}}{\delta t}, \lambda_z > \tag{29}$$

Sin embargo, para este proyecto se revisó y aplicó la propiedad de simetría que implica el uso de operadores lineales tal como sigue: Sea A cualquier operador lineal (e.g. $\frac{\partial}{\partial t}$) y los elementos x y y que pertenecen al espacio V entonces

$$< Ax, y > = < x, A^* y > \tag{30}$$

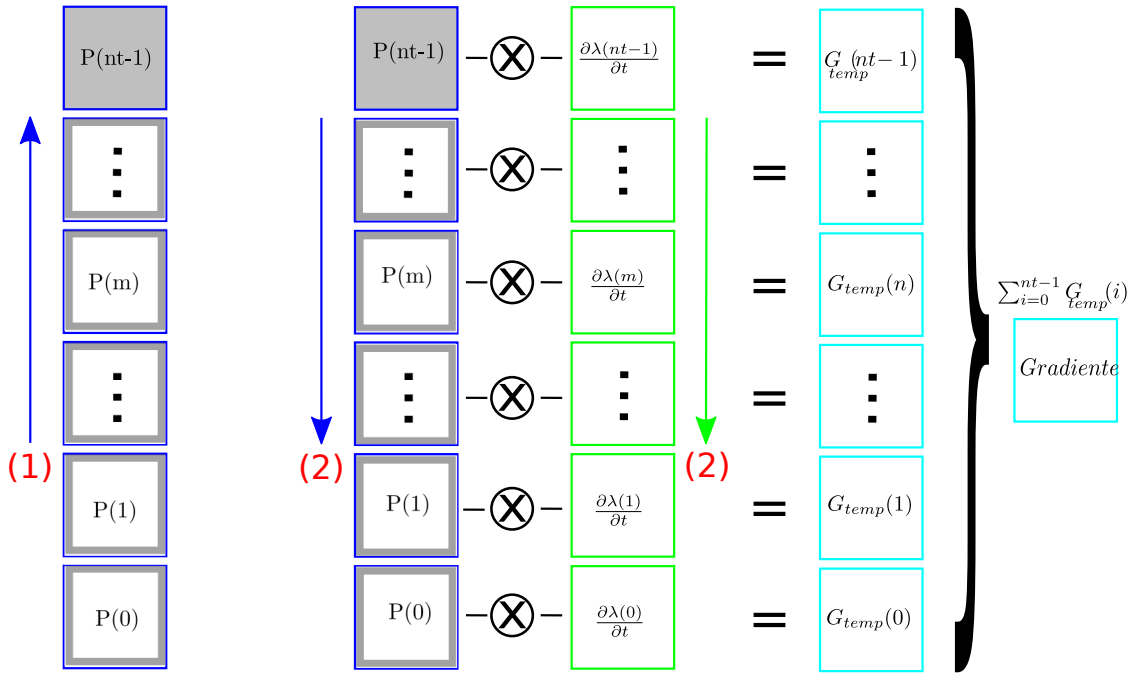
donde A^* es el adjunto de A. Con ayuda de esta propiedad de simetría, es posible reescribir las expresiones 28 y 29 como las ecuaciones 31 y 32, respectivamente.

$$G_{\mathbf{c}} = \frac{-2}{\rho \mathbf{c}^3} < \mathbf{P}, -\frac{\partial \lambda}{\partial t} > \tag{31}$$

$$G_{\rho} = \frac{-1}{\rho^2 \mathbf{c}^2} < \mathbf{P}, -\frac{\partial \lambda}{\partial t} > + < \mathbf{V_x}, -\frac{\partial \lambda_x}{\delta t} > + < \mathbf{V_y}, -\frac{\partial \lambda_y}{\delta t} > + < \mathbf{V_z}, -\frac{\partial \lambda_z}{\delta t} > \tag{32}$$

Teniendo esto en cuenta, la estrategia de reconstrucción de campo que se grafica en la figura 29 ahora se convierte en el descrito por la figura 30. Gracias a la propiedad de simetría de producto intero se disminuye la cantidad de etapas de modelado y la reducción de memoria se mantiene en el mismo factor.

Figura 30. Proceso de reconstrucción de campo cuando se aprovecha la propiedad de simetría del producto interno. Nótese que ahora se reconstruye la información asociada al campo de propagación y solo se necesitan dos etapas de simulación lo cuál va a reducir el tiempo de ejecución del método de inversión.



5.4. Resumen general de la estrategia definida

Teniendo en cuenta que, además de los ítems definidos al inicio del capítulo, se encontraron características de la GPU útiles para generar un alto rendimiento de la implementación desarrollada y una propiedad matemática que permite reducir la cantidad de operaciones necesarias para obtener el gradiente mediante la metodología de reconstrucción de campo. A continuación se presenta el esquema que resume la estrategia definida con las nuevas características (ver figura 31):

- Metodología base: Se usa una metodología de reconstrucción de campo para obtener el gradiente y ahorrar el consumo de memoria en más de un $90\,\%$. Se debe tener en cuenta que para aplicar el método de reconstrucción se debe trabajar con una ecuación de onda que no contenga parámetros de atenuación en la expresión o expresiones que la conforman[56].

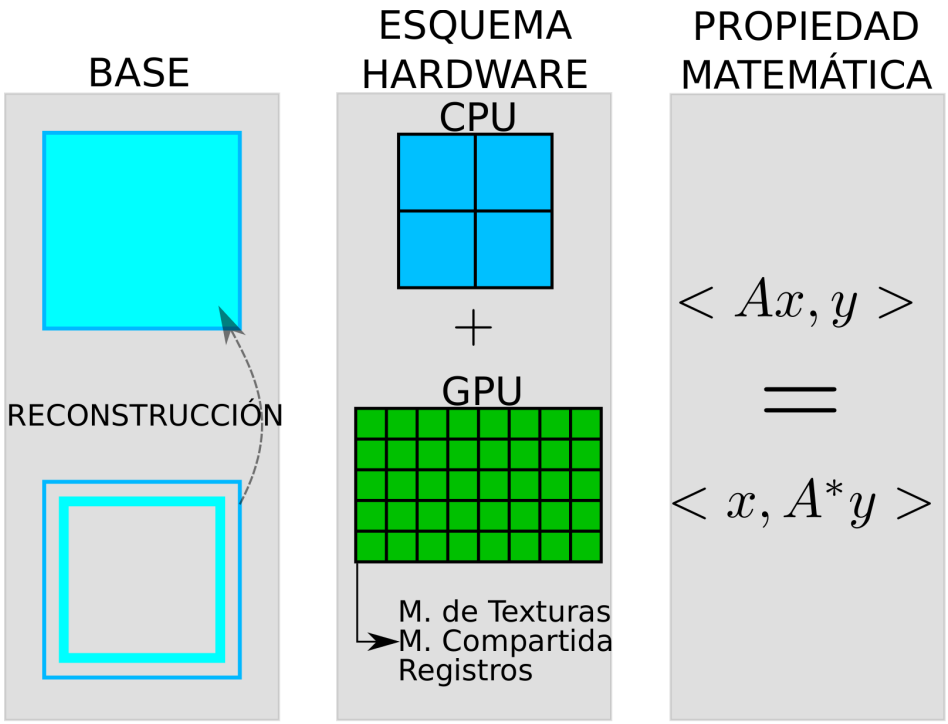
- Revisión de hardware de cómputo: Se analizó el sistema de trabajo como un conjunto de paresjas de CPUs y GPUs (que se ve a gran escala en los supercomputadores a nivel mundial). Se aprovechó la capacidad de acceso de la CPU a una alto espacio de memoria RAM y el parelismo de las GPUs para lidiar con los procesos de mayor complejidad computacional. A nivel de disco duro se accede al dato adquirido debido a que su lectura desde la RAM principal no se considera conveniente debido al gran volumen de información. Adicionalmente, se revisó la arquitectura de GPU para compensar la penalización al tiempo de ejecución que genera la metodología de reconstrucción , las trasnferencias de datos entre RAM de CPU y GPU y los accesos de disco

[56] Los actuales resultados de investigación de un proyecto de doctorado dentro del grupo de investigación CPS han demostrado que los parámetros de atenuación eliminan la posibilidad de reconstruir el campo de propagación debido a que la información del frente de onda disminuye rápidamente de amplitud y no alcanza las fronteras del modelo.

duro.

- Propiedad matemática: Se aprovechó la propiedad de simetría del producto interno para obetener el mismo gradiente con el mismo porcentaje de ahorro de memoria y realizando menor cantidad de operaciones. Esta propiedad se complementa con lo mencionado en el ítem anterior para compensar las penalizaciones del tiempo de ejecución.

Figura 31. Resumen de la estrategia definida. 1) Se usa la metología de reconstrucción de campo por su gran beneficio en cuanto al ahorro de memoria. 2) Un análisis general a nivel de la arquitectura del cluster, el uso de los APIs escogidos a nivel de CPU y GPU se usan para distribuir de forma balanceada los procesos del algoritmo de FWI en los procesadores de acuerto a sus característica de rendimiento. 3) Una revisión de la arquitectura de GPU



5.5. Medidas de rendimiento dentro del cluster

5.5.1. Memoria utilizada Para conocer las limitaciones de la estrategia computacional propuesta en este trabajo se plantean las expresiones (33) y (34), las cuales indican la cantidad de memoria a usar (en $gigabytes$) por cada procesador (GPU y CPU). R_{GBGPU} hace referencia a la memoria disponible en la GPU y R_{GBCPU} hace referencia a la memoria disponible en la CPU. El factor de escala 23 es debido a la cantidad de volúmenes que se almacenan en la memoria de la GPU para realizar las respectivas etapas de propagación, la variable L_{BFGS} que hace referencia a la cantidad de volúmenes de modelos y gradientes que se deben guardar según el método L-BFGS.

$$R_{GBGPU} \geq \frac{4}{1 \times 10^9}(23 \times N_x \times N_y \times N_z + 3 \times N_{rx} \times N_{ry}) \tag{33}$$

$$\begin{aligned} R_{GBCPU} \geq \frac{4}{1 \times 10^9}(2 \times s \times (N_x \times N_y + N_x \times N_z + N_y \times N_z) \times N_t \\ + 3 \times N_{rx} \times N_{ry} \times N_t + (4 \times L_{BFGS} + 7) \times N_x \times N_y \times N_z) \end{aligned} \tag{34}$$

Se debe asegurar que ambas relaciones se cumplan al momento seleccionar las dimensiones del modelo a procesar para no caer en problemas como uso de la memoria swap[57] o errores de reserva de RAM.

---

[57] Espacio en disco duro que se utiliza cuando el sistema operativo detecta que no hay RAM suficiente para seguir almacenando datos. Puede considerarse como un espacio de emergencia de uso automático pero que afectará el rendimiento del sistema.

5.5.2. Tiempo de ejecución El tiempo de ejecución (en días) se puede estimar, en un escenario ideal[58] con la expresión (35).

$$T_{FWI3D} = \frac{5 \times t_{prop} \times N_{iter} \times N_{src}}{N_{GPUs}} \tag{35}$$

Donde N_{GPUs} es la cantidad de GPUs que se están utilizando al momento de la ejecución, t_{prop} el tiempo de ejecución de una propagación en una sola GPU, el factor 5 hace referencia a las cuatro etapas de propagación (propagación, retropropagación, reconstrucción y propagación en los intentos del método L-BFGS) implicadas en cada iteración del método más el tiempo requerido para las continuas transferencias entre memorias (host y device) y procesos en CPU, N_{iter} es la cantidad de iteraciones de FWI que se quieren y N_{src} es la cantidad de fuentes utilizadas al momento de la ejecución del algoritmo.

5.5.3. Porcentaje de utilización de la GPU Durante la ejecución de aplicaciones es necesario que los procesos realizados en CPU, cuya salida son entradas de $kernels$ en la GPU, sean de corta duración. De no ser así, la GPU queda en un estado de no actividad, o cero utilización, medido como un porcentaje de trabajo en tiempo de ejecución. Con ayuda de la la herramienta $nvidia - smi$[59] se puede medir la carga que se hace sobre la GPU durante la ejecución de la implementación. Esta medida es importante para verificar que la mayor parte del proceso se está realizando en la GPU donde un valor de $100\,\%$ indica que las transferencias de datos y los procesos en CPU no tienen un efecto negativo significativo en el rendimiento

---

[58] El método de L-BFGS implica una propagación adicional por cada intento de obtener el mejor paso de avance en cada iteración de FWI. Idealmente debería encontar este paso con un solo intento en cada iteración del método de inversión.

[59] Comando en sistema operativo Linux para revisar las aplicaciones que se están ejecutando en la GPU.

de la aplicación.

5.5.4. FLOPS y ancho de banda Del inglés Floating Point Operations per Second, la medida de FLOPs indica cuántas operaciones de punto flotante se realizan por segundo en la aplicación. Si un $kernel$ se implementa de tal forma que solo realiza operaciones aritméticas de punto flotante, se considera como $kernel$ limitado por cómputo[60] (el desempeño incrementa si el hardware sobre el que se ejecuta la aplicación es mejor en términos de frecuencia de trabajo o cantidad de recursos disponibles por núcleo de procesamiento.).
Por otro lado, el ancho de banda de memoria es otra medida de rendimiento bastante común en implementaciones en CUDA. Si el $kernel$ está enfocado principalmente en accesos a memoria, se dice que es un $kernel$ limitado por memoria[61] (el desempeño incrementa si el hardware es mejor en términos del ancho de bus de transferencias de datos y mayor capacidad en los distintos niveles de memoria). Sin embargo, es común que los $kernels$ implementados en cualquier aplicación (incluyendo la implementación del método FWI 3D desarrollada para este trabajo) se compongan de una mezcla de operaciones aritméticas y operaciones de memoria. Esto implica que se debe encontrar un balance tanto en ancho de banda como en FLOPS de cada $kernel$ implementado para extraer el mayor desempeño del dispositivo.

---

[60] $compute\ bound$ en inglés.

[61] $memory\ bound$ en inglés.

6. RESULTADOS

6.1. Rendimiento

De acuerdo a las mejoras mencionadas en el capítulo anterior y un análisis iterativo usando la herramienta $Nvidia\ Visual\ Profiler$, fue necesario modificar la función de propagación y crear más $kernels$ de $CUDA$. Ahora esta función se compone de los $kernels$ descritos a continuación:

- `kernelP();`: Kernel usado para calcular el campo P, en cada muestra de tiempo, a partir de las derivadas en las direcciones x, y y z de de los campos V_x, V_y y V_z, respectivamente.
- `kernelVelX();`: Kernel usado para calcular, en cada muestra de tiempo, el campo V_x a partir de las derivadas en la dirección x del campo$P$.
- `kernelVelY();`: Kernel usado para calcular, en cada muestra de tiempo, el campo V_y a partir de las derivadas en la dirección y del campo$P$.
- `kernelVelZ();`: Kernel usado para calcular, en cada muestra de tiempo, el campo V_z a partir de las derivadas en la dirección z del campo$P$.

Además de la creación de más $kernels$, se tienen las medidas mostradas en la tabla 4.

---

[1] Teniendo en cuenta que las dimensiones del modelo usado son de $250\times250\times140$ y $256\times250\times140$ puntos para la primera y segunda versión de la función de propagación, respectivamente.

[2] La herramienta $Nvidia\ Visual\ Profiler$ califica el uso de cada tipo de memoria en valores de 0 (ningún uso del recurso) a 9 (uso máximo del recurso).

Tabla 4. Primer comparación entre los $kernels$ implementados para las dos versiones de la función de propagación[1].

Kernel	Transacciones de memoria	Uso de memoria de texturas[2]	Configuración de Grid	Configuración de bloque
kP_1	4.7472	0	$3D$	$3D$
kV_{xyz}	4.7053	0	$3D$	$3D$
kP_2	1.0000	6	$2D$	$2D$
kV_x	1.0000	5	$2D$	$2D$
kV_y	1.0000	5	$2D$	$2D$
kV_z	1.0000	0	$3D$	$3D$

Donde los valores de kP_1 y kV_{xyz} hacen referencia a los $kernels$ `kernelP();` y `kernelVelXYZ();`, respectivamente, de la primer versión de la función de propagación. Los valores de kP_1, kV_x, kV_y y kV_z hacen referencia a los $kernels$ `kernelP();` y `kernelVelX();`, `kernelVelY();` y `kernelVelZ();`, respectivamente, de la segunda versión de la función de propagación.

Como ya se mencionó, una de las dos formas más comúnes de medir el rendimiento es calcular la cantidad de $FLOPs$ de cada $kernel$ implementado. La figura 32 muestra la comparación de rendimiento (en precisión simple) de los $kernels$ que componen la primera y segunda versión de la función de propagación.

Por otro lado, la medida del ancho de banda de cada $kernel$ indica el uso de la GPU en cuanto a la cantidad de operaciones de memoria realizadas. La figura 33 muestra el ancho de banda de la RAM de la GPU para los $kernels$ de la primera y segunda versión de la función de propagación.

Nótese que las medidas de ancho de banda se ubican en un rango alrededor del $60\,\%$ del valor máximo ($288[GB/s]$). Aunque la herramienta de análisis marca un umbral de rendimiento aceptable por encima de éste número, no se alcanza el máximo rendimiento debido a los multiples accesos que deben hacer los $threads$ a diferentes bloques de memoria asociado a las diferentes variables de campos de presión y campos de velocidad. Por otro lado, el valor más grande de FLOPs medidos fue

Figura 32. Comparación del rendimiento en términos de GFLOPs de los $kernels$ de la función de propagación antes y después de aplicar las mejoras.

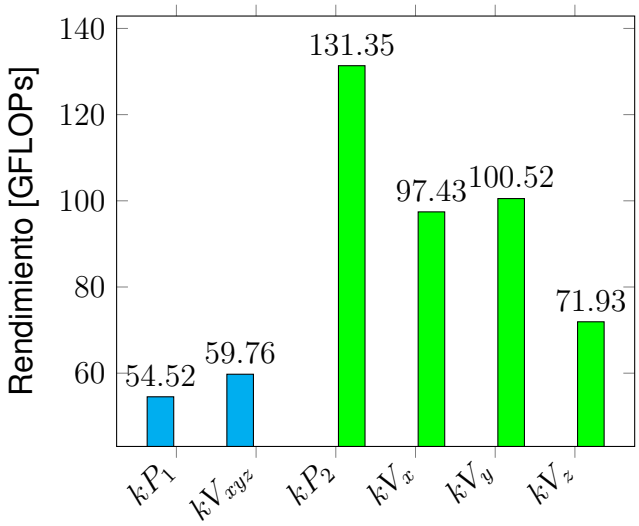
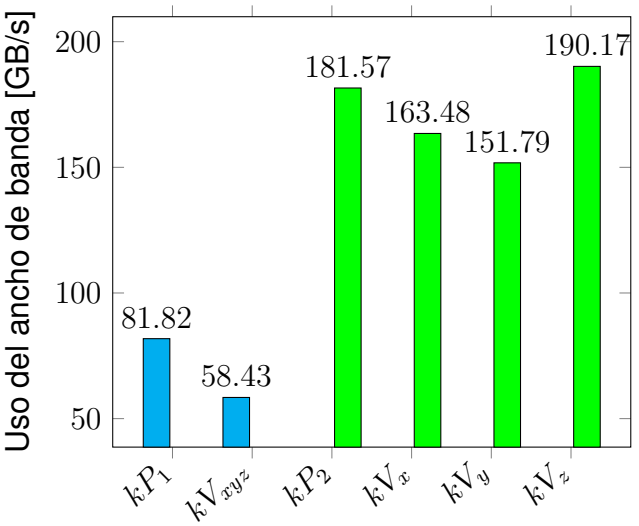


Figura 33. Comparación del ancho de banda de RAM usado por los $kernels$ de la función de propagación antes y después de aplicar las mejoras.



de $131.3533[GFLOPs]$. Teniendo en cuenta que la $GPU\ TESLA\ K40$ tiene un pico máximo teórico de rendimiento de $4.29[TFLOPs]$, se revisaron los aspectos que impiden que las medidas alcancen un valor cercano a éste número.

6.1.1. Paralelismo a nivel de instrucción Del inglés *Instruction Level Parallelism*, el ILP hace referencia al trabajo realizado por cada $thread$, de esta forma, entre mayor cantidad de instrucciones se estén ejecutando por hilo, mayor será la medida de $FLOPs$ obtenida. Para comprobar esto se comparan los cuatro $kernels$ de las listas 6.1, 6.2, 6.3 y 6.4, obteniendo los resultados mostrados en la tabla 5.

Listing 6.1. Kernel 1. Dentro de un bucle se actualiza una variable en registros y este valor se asigna a una posición de un cubo x en RAM.

```
__global__
void k1(float *x){
...
//tid es una variable asociada a cada thread con una posicion
//de memoria diferente dentro del cubo x
float a=1.0, b=1.0;
for(int i=0;i<4096;i++){
  a = a*b + b;
  }
x[tid]=a;
}
```

Listing 6.2. Kernel 2. Dentro de un bucle se actualiza una variable en registros a partir de la informacion contenida en un cubo d en RAM. Luego este valor se asigna a una posición de un cubo x en RAM.

```
__global__
void k2(float *x, float *d){
...
//tid es una variable asociada a cada thread con una posicion
//de memoria diferente dentro de los cubos x y d
float a=1.0;
for(int i=0;i<4096;i++){
  a = a*(d[tid] + d[tid+1] + d[tid-1] + d[tid+2] + d[tid-2]);
  }
x[tid]=a;
}
```

Listing 6.3. Kernel 3. Se actualiza una variable en registros a partir de la informacion contenida en un cubo d en RAM.

```
__global__
void k3(float *d){
...
//tid es una variable asociada a cada thread con una posicion
//de memoria diferente dentro del cubo d
float a=1.0;
a = a*(d[tid] + d[tid+1] + d[tid-1] + d[tid+2] + d[tid-2]);
}
```

Listing 6.4. Kernel 4. Se actualiza la informacion del cubo x con la informacion del cubo d.

```
__global__
void k4(float *d){
...
//tid es una variable asociada a cada thread con una posicion
//de memoria diferente dentro del cubo d
float a=1.0;
x[tid] = x[tid] + a*(d[tid] + d[tid+1] + d[tid-1] + d[tid+2] + d[tid-2]);
}
```

Tabla 5. Comparación de rendimiento de cuatro kernels al cambiar el ILP y la cantidad de operaciones de memoria por $thread$ ejecutado.

kernel	Rendimiento [GFLOPs]	Ancho de banda [GB/s]
k1	3.10×10^3	1.8
k2	1.71×10^3	2.2
k3	55.86	23.48
k4	51.70	20.22

Nótese que en los $kernels$ $k1$ y $k2$ la cantidad de operaciones aritméticas es mucho mayor (bucle) que la cantidad de operaciones de acceso a memoria que debe realizar cada $thread$ y por lo tanto pueden extraer un mayor rendimiento en cuanto a cantidad de $FLOPs$ ($compute$ $bound$). Por otro lado, en los $kernels$ $k3$ y $k4$ la cantidad de operaciones aritméticas no es tan alta y el rendimiento está limitado por las operaciones de acceso a memoria ($memory$ $bound$). Teniendo estos resultados en cuenta y la similitud entre el $kernel$ $k4$ y los $kernels$ implementados para la función propagación, los valores de rendimiento mostrados en la figura 32 se consideran como aceptables debido a la naturaleza del algoritmo ($memory$ $bound$).

6.1.2. Desaceleración con memoria compartida Aunque en el capítulo anterior se mencionó la utilidad de la memoria compartida en aplicaciones creadas con CUDA, no siempre se recomienda su uso para incrementar el rendimiento de un $kernel$ [62]. La instalación de $CUDA$ incluye un ejemplo del uso de memoria compartida para la ecuación de onda acústica con densidad constante[63]. A continuación se explican las diferencias con el operador usado en el presente trabajo:

- Derivada espacial: Con la ecuación de onda acústica con densidad constante se debe acceder al bloque de memoria usado para almacenar $\mathbf{P}$ y aplicar la derivada espacial en las tres dimensiones. Sin embargo, para el caso del operador con densidad variable se deben acceder a los espacios de memoria que almacenan la informacion de $\mathbf{V_x}$, $\mathbf{V_y}$ y $\mathbf{V_z}$ para aplicar las respectivas derivadas espaciales.

- Cantidad de variables: Mientras que con el operador de onda acústico con densidad constante se debe acceder a la información de $\mathbf{P}$ y $\mathbf{c}$ (almacenada en memoria global), para el caso de la ecuación con densidad variable se debe acceder a la información de $\mathbf{P}$, $\mathbf{V_x}$, $\mathbf{V_y}$, $\mathbf{V_z}$, $\mathbf{c}$ y ρ.

Las diferencias mencionadas implican que, para el caso de la ecuación con densidad variable, dentro de los $kernels$ se deben realizar más accesos a memoria global (con su respectiva copia a memoria compartida) para realizar las derivadas espaciales. Esto significa que habrá una penalización de acceso a memoria global (que incrementa con el orden de aproximación espacial de diferencias finitas utilizado)

---

[62] Vasily Volkov y James W Demmel. “Benchmarking GPUs to tune dense linear algebra”. En: *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*. IEEE. 2008, págs. 1-11.

[63] El kernel implementado utiliza diferencias finitas con aproximación de segundo orden temporal, segundo orden espacial y tiene un rendimiento de $160[GFLOPs]$, aproximadamente

que reducirá el rendimiento de la función implementada.

El uso de memoria compartida disminuyó el rendimiento de la función propagación en $10\,\%$, aproximadamente. Teniendo en cuenta lo anteriormente mencionado se decide no hacer uso de la memoria compartida dentro de la implementación.

6.1.3. Factor de aceleración No alcanzar el valor máximo de FLOPs, o no poder hacer uso de la memoria compartida, no equivale a un mal rendimiento. En la tabla 6 se muestran los tiempos de ejecución para los mismos tres experimentos con los que se obtuvo la tabla 2. Los resultados de esta tabla indican que el factor de acele-

Tabla 6. Tiempo de ejecución de segunda versión de la función de propagación para los tres experimentos con los que se obtuvo la tabla 2. La penalización de tiempo es relativa al experimento 1.

Experimento	Tiempo de ejecución [s]	Penalización de tiempo [$\%$]	Bytes escritos
1	19.31	n/a	n/a
2	19.93	3.21	23.04[MB]
3	349.20	1708.38	280[GB]

ración de la segunda versión de la función de propagación respecto a la primera es de 3.50x, aproximadamente.

6.2. Resultados FWI 3D

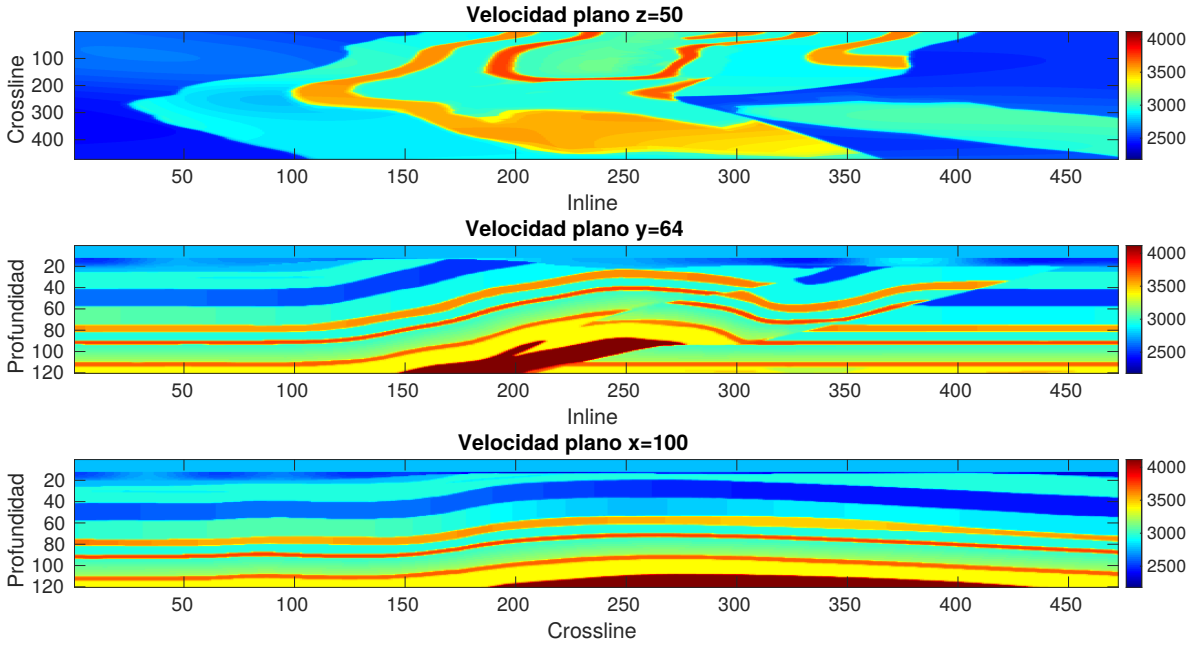
6.2.1. Modelo Overthrust 3D Se utilizó como primer modelo original, para la generación del dato observado, el modelo de velocidades overthrust [64] que se aprecia en la figura 34 y el modelo de densidades se obtuvo mediante la relación de brocher [65] (ver figura 35).

---

[64] JC Lecomte, E Campbell y J Letouzey. "Building the SEG/EAEG overthrust velocity macro model". En: *EAEG/SEG Summer Workshop-Construction of 3-D Macro Velocity-Depth Models*. 1994.

[65] Thomas M Brocher. "Empirical relations between elastic wavespeeds and density in the Earth's crust". En: *Bulletin of the seismological Society of America* 95.6 (2005), págs. 2081-2092.

Figura 34. Tres cortes del modelo real de velocidades $Overthrust3D$ usado para generar el dato observado. Sólo se grafica la zona de interés, despreciando la información dentro del área de CPML.



Según las ecuaciones 33 y 34 y el sistema de cómputo disponible, el modelo más grande que se puede procesar debe tener dimensiones $512 \times 512 \times 140$ con una fuente de 5000 muestras de tiempo. Por cada fuente procesada dentro de cada nodo de cómputo, el uso de RAM de GPU se estima en $7.37[GB]$ y el uso de RAM de CPU en $85.14[GB]$. Teniendo esto en cuenta, se plantea un experimento con la inicialización de parámetros que se aprecia en la tabla 7 y el tiempo de ejecución estimado es de 4.16 días.

Evaluando la ecuación 21, se tiene que la inicialización de parámetros cumple con el criterio de $Courant$ y la cantidad de puntos por longitud de onda, según la ecuacion 22 es de aproximadamente 6. Nótese que si la frecuencia de la fuente se incrementa a $6[Hz]$ (valor de ejemplo si se desea realizar un proceso multi-escala), la cantidad de puntos por longitud de onda disminuye a 3 generando dispersión numérica. Para

Figura 35. Tres cortes del modelo real de densidades $Overthrust3D$ usado para generar el dato observado. Sólo se grafica la zona de interés, despreciando la información dentro del área de CPML.

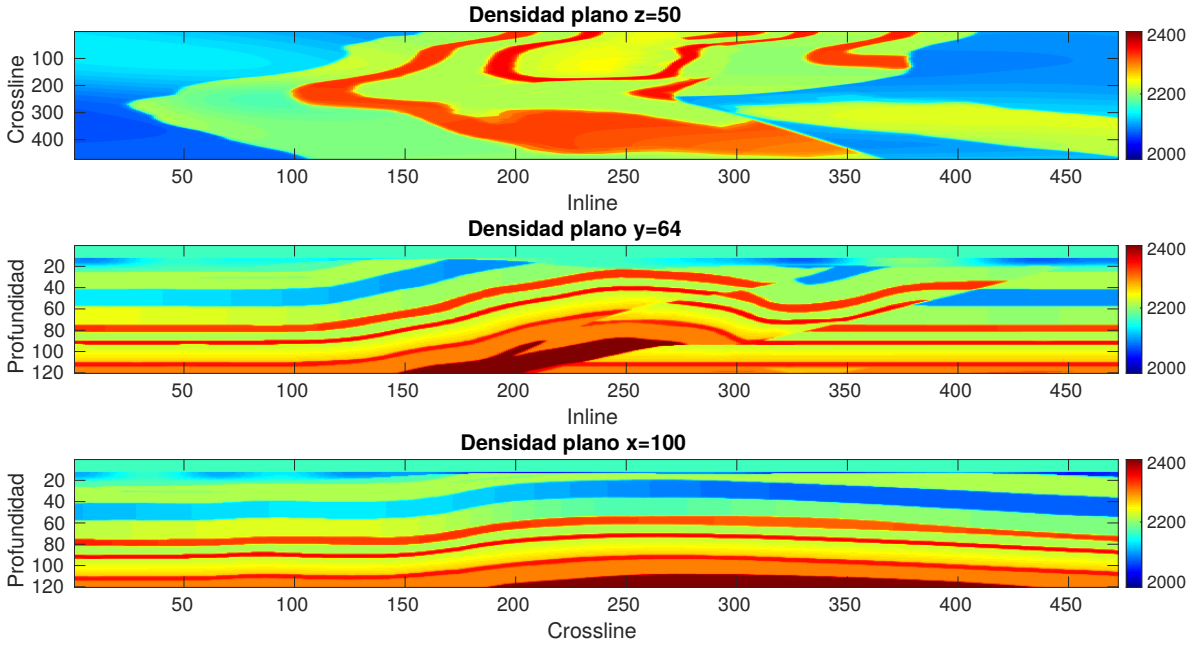


Tabla 7. Inicialización de parámetros para los experimentos de FWI 3D.

Variable	Valor	Descripción
N_x	512	Cantidad de puntos en el eje x
N_y	512	Cantidad de puntos en el eje y
N_z	140	Cantidad de puntos en el eje z
N_t	5000	Cantidad de muestras de tiempo
Δ_x	$50[m]$	Paso espacial en el eje x
Δ_y	$50[m]$	Paso espacial en el eje y
Δ_z	$50[m]$	Paso espacial en el eje z
Δ_t	$1[ms]$	Paso temporal
$wcpml$	20	Ancho en puntos de la fontera CPML
f	$3[Hz]$	Frecuencia central de la fuente
$nRecs$	472×472	Cantidad de receptores en superficie
$nIter$	20	Cantidad de iteraciones de FWI
nS	121	Cantidad de fuentes

evitar esto, se deben remuestrear los modelos de velocidad y densidad para obtener un paso espacial de $25[m]$ en cada dimensión, obteniendo cubos de $1024 \times 1024 \times 280$ elementos. Con estas nuevas dimensiones se tendría un uso de $27.10[GB]$ de memoria de GPU y $372.81[GB]$ de memoria de CPU por cada fuente procesada al interior de cada nodo de cómputo, excediendo los límites de recursos disponibles del clúster $Tokyo$.

Los modelos originales se suavizan y se obtienen los puntos iniciales que se aprecian en las figuras 36 y 38.

Después de 4.5 días los modelos de velocidad y densidad obtenidos son los que se aprecian en las figuras 37 y 39, respectivamente. La memoria usada de la GPU durante la ejecución de la FWI fue de $7.4[GB]$. En cuanto al uso de memoria de CPU se puede apreciar con la herramienta $htop$ de $Linux$ que el uso está alrededor de $85[GB]$.

Figura 36. Tres cortes del modelo inicial de velocidades Overthrust 3D usado como punto de partidad de la FWI.

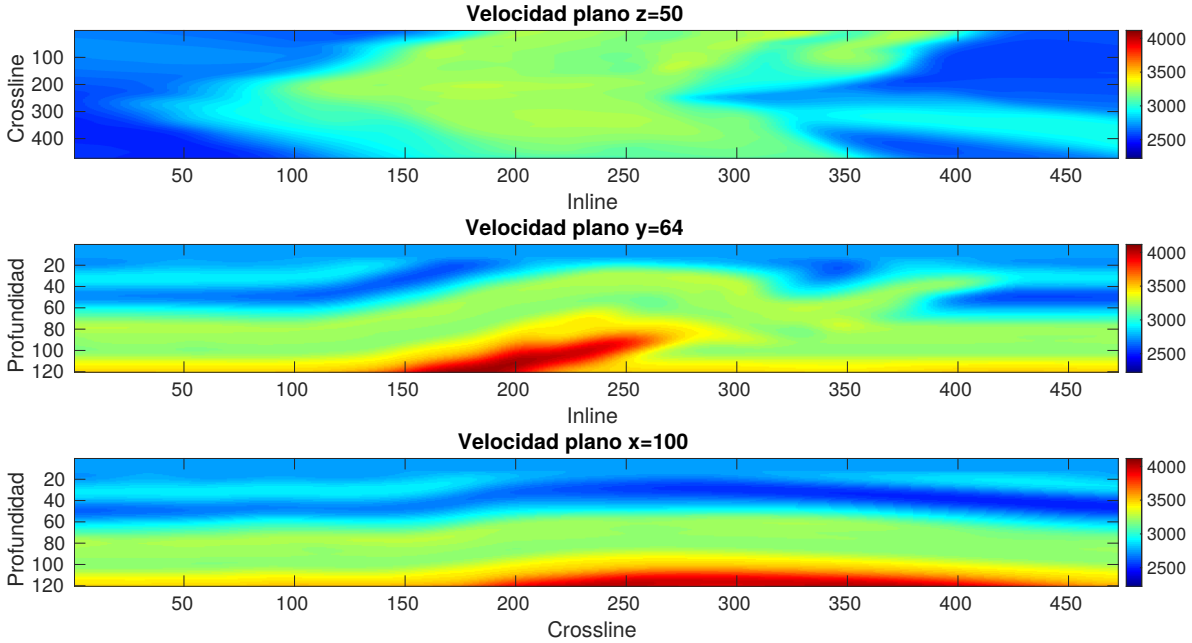


Figura 37. Tres cortes del modelo final de velocidades $Overthrust3D$.

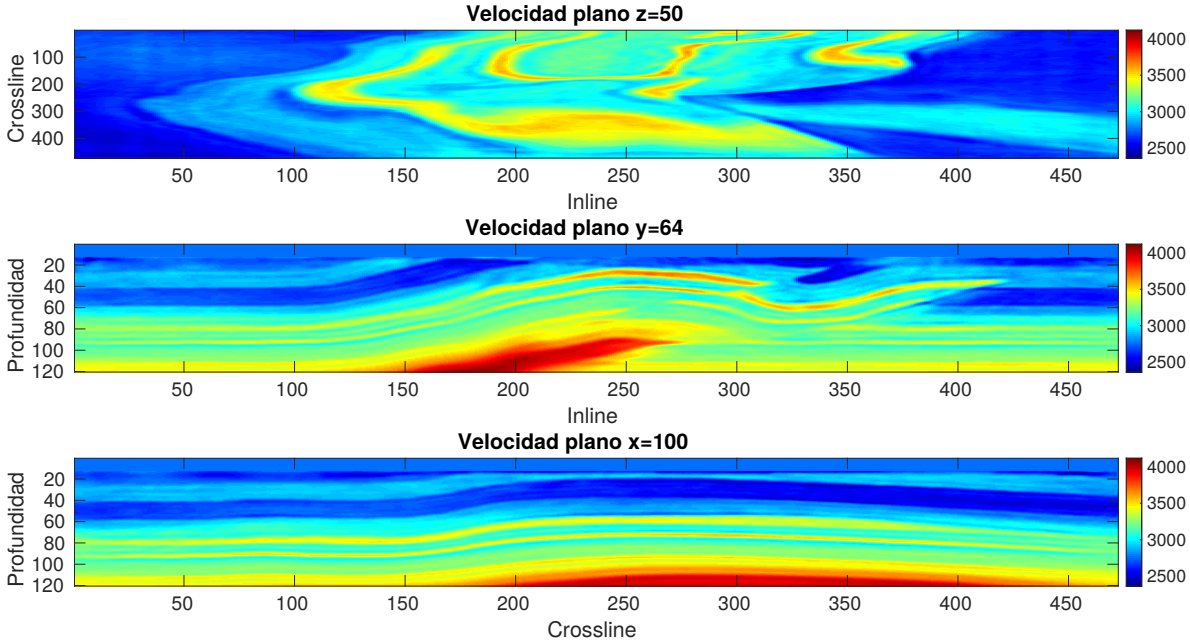


Figura 38. Tres cortes del modelo inicial de densidades $Overthrust3D$ usado como punto de partida de la FWI.

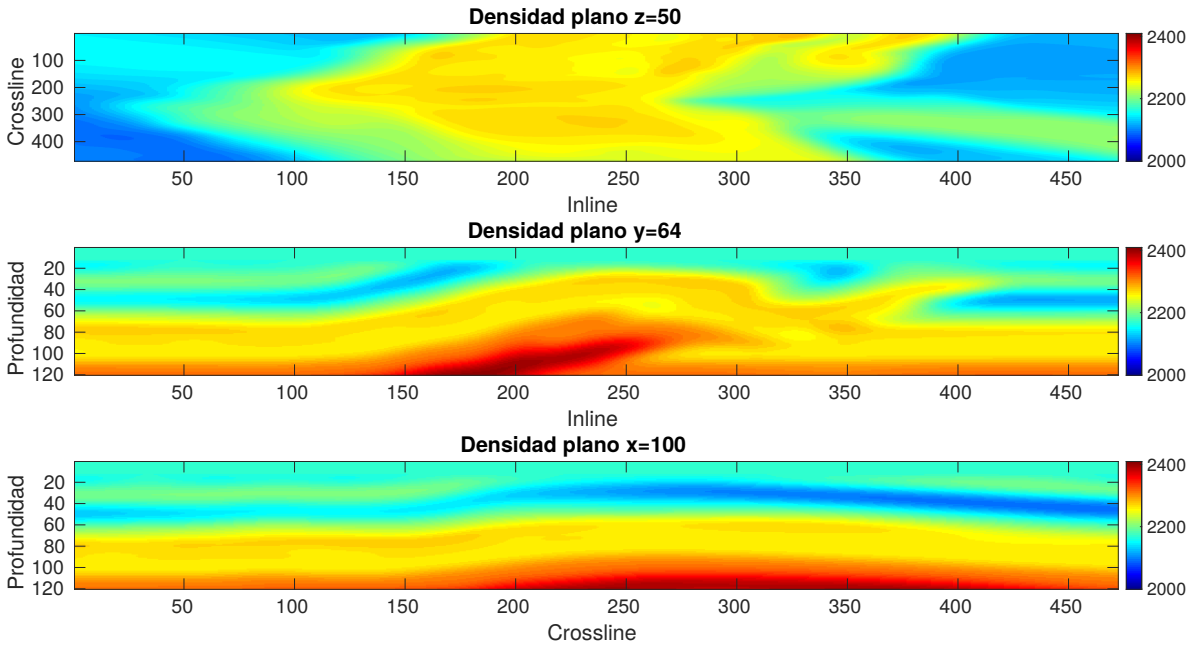
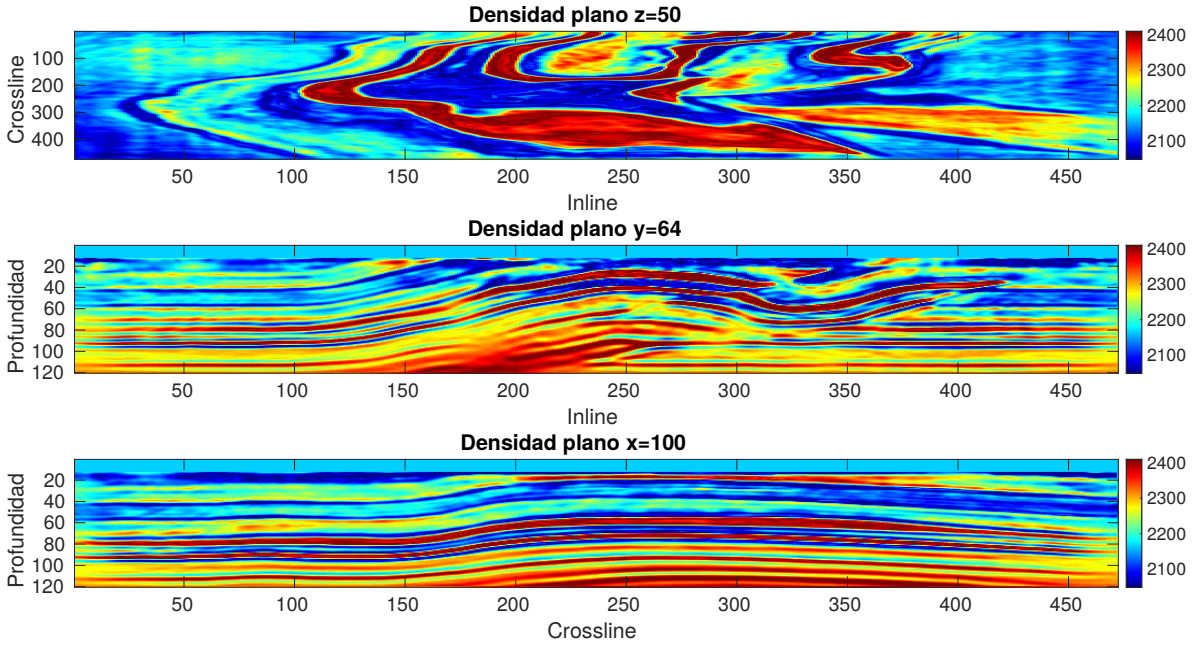


Figura 39. Tres cortes del modelo final de densidades $Overthrust3D$.



6.2.2. Modelo SEAM 3D Gracias a un convenio existente con la Empresa Colombiana de Petróleos - ECOPETROL[66], se tuvo acceso a una porción de un modelo de velocidades, con su respectiva información de densidades, conocido como $SEAM\ 3D$. La inicialización de parámetros para este experimento se muestra en la tabla 8. A partir de las ecuaciones propuestas, la RAM usada, por cada fuente procesada al tiempo dentro de cada nodo, será de $912[MB]$ en la GPU y $14.22[GB]$ en la CPU. Adicionalmente, el tiempo de ejecución se estima en 9.38 horas.

Tabla 8. Inicialización de parámetros para los experimentos de FWI 3D.

Variable	Valor	Descripción
N_x	384	Cantidad de puntos en el eje x
N_y	157	Cantidad de puntos en el eje y
N_z	150	Cantidad de puntos en el eje z
N_t	2500	Cantidad de muestras de tiempo
Δ_x	$40[m]$	Paso espacial en el eje x
Δ_y	$40[m]$	Paso espacial en el eje y
Δ_z	$40[m]$	Paso espacial en el eje z
Δ_t	$2[ms]$	Paso temporal
$wcpml$	20	Ancho en puntos de la fontera CPML
f	$3[Hz]$	Frecuencia central de la fuente
$nRecs$	344×117	Cantidad de receptores en superficie
$nIter$	20	Cantidad de iteraciones de FWI
nS	48	Cantidad de fuentes

Los modelos reales de velocidades y densidades (usados para generar el dato observado) se aprecian en las figuras 40 y 41, respectivamente y los respectivos puntos de partida para la $FWI\ 3D$ se aprecian en las figuras 42 y 44.

Después de 10.55 horas, aproximadamente, los modelos finales de velocidad y de densidad son los que se muestran en las figuras 43 y 45, respectivamente. El uso de memoria de GPU fue de $837[MiB]$ y aproximadamente $14[GB]$ en CPU.

---

[66] Este trabajo tiene el apoyo de la Emepresa Colombiana de Petróleos bajo el convenio No. 0266 del 2013.

Figura 40. Tres cortes del modelo original de velocidades $SEAM\ 3D$.

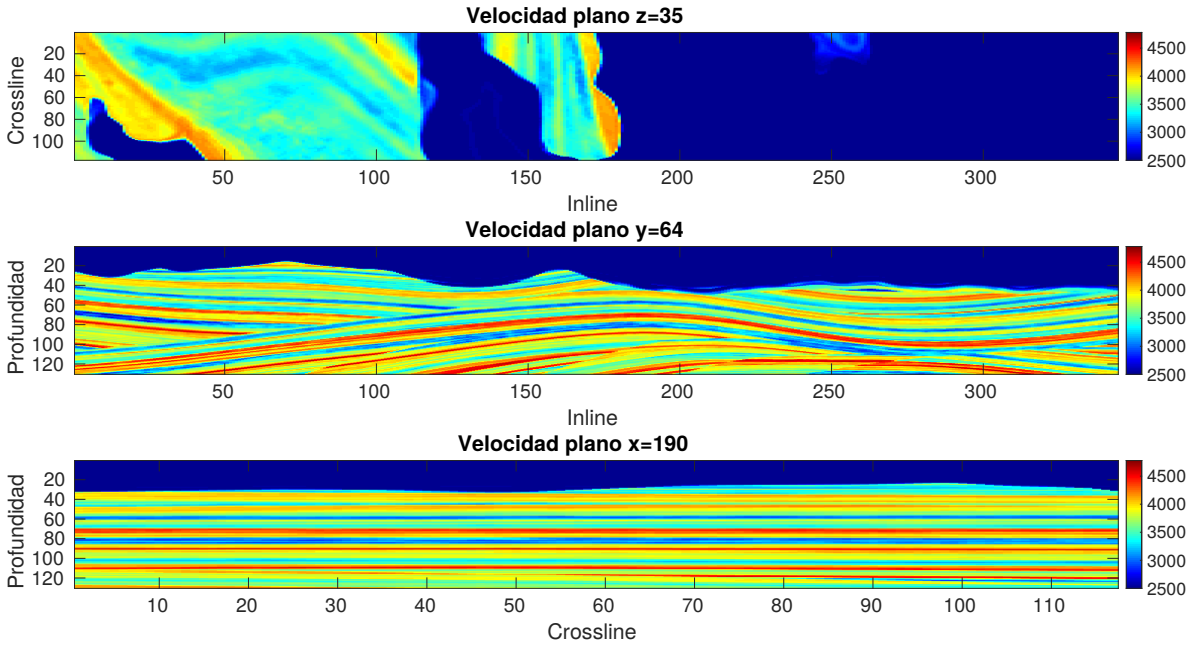


Figura 41. Tres cortes del modelo original de densidades $SEAM\ 3D$.

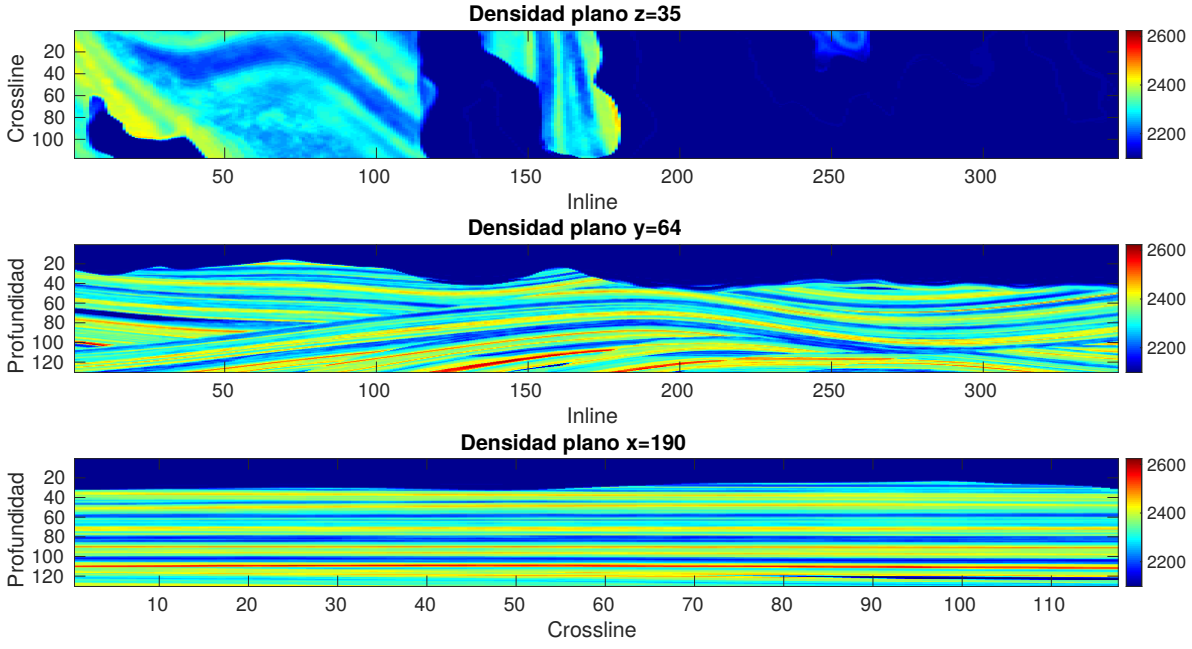


Figura 42. Tres cortes del modelo inicial de velocidades $SEAM\ 3D$.

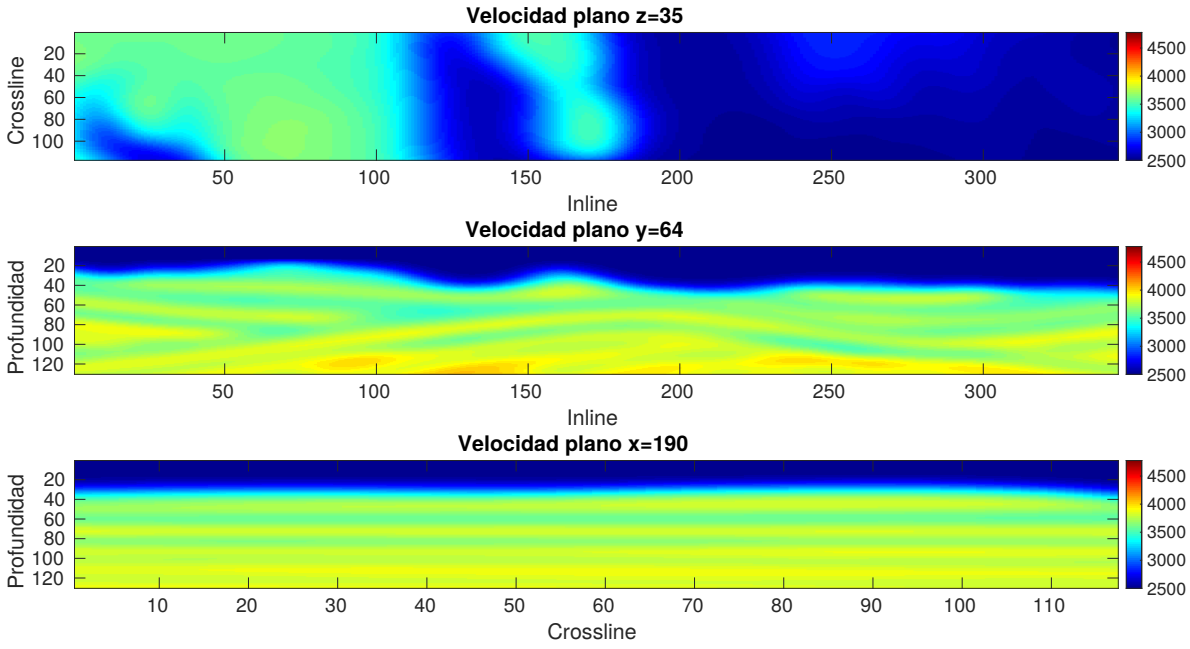


Figura 43. Tres cortes del modelo final de velocidades $SEAM\ 3D$.

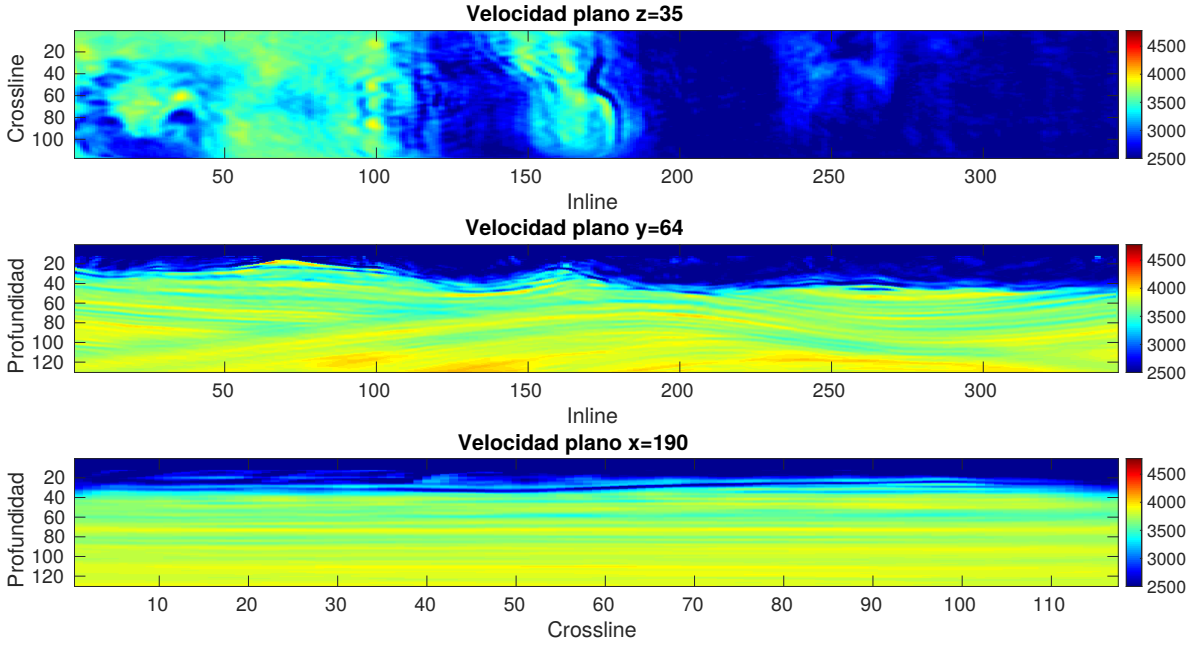


Figura 44. Tres cortes del modelo inicial de densidades $SEAM\ 3D$.

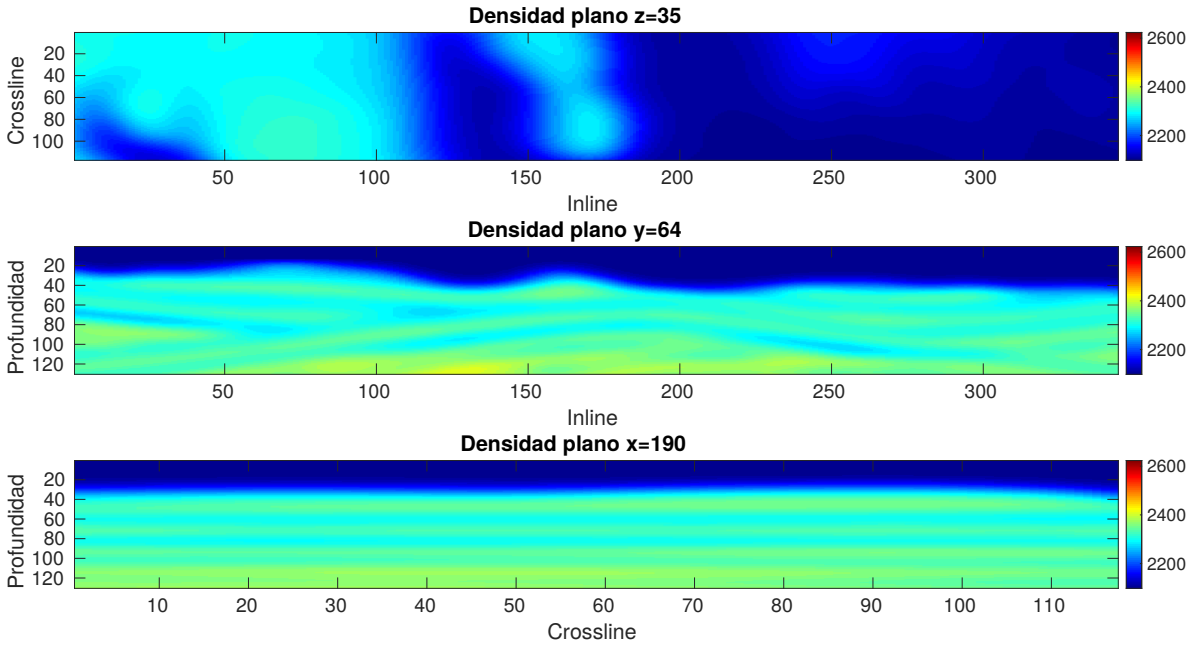
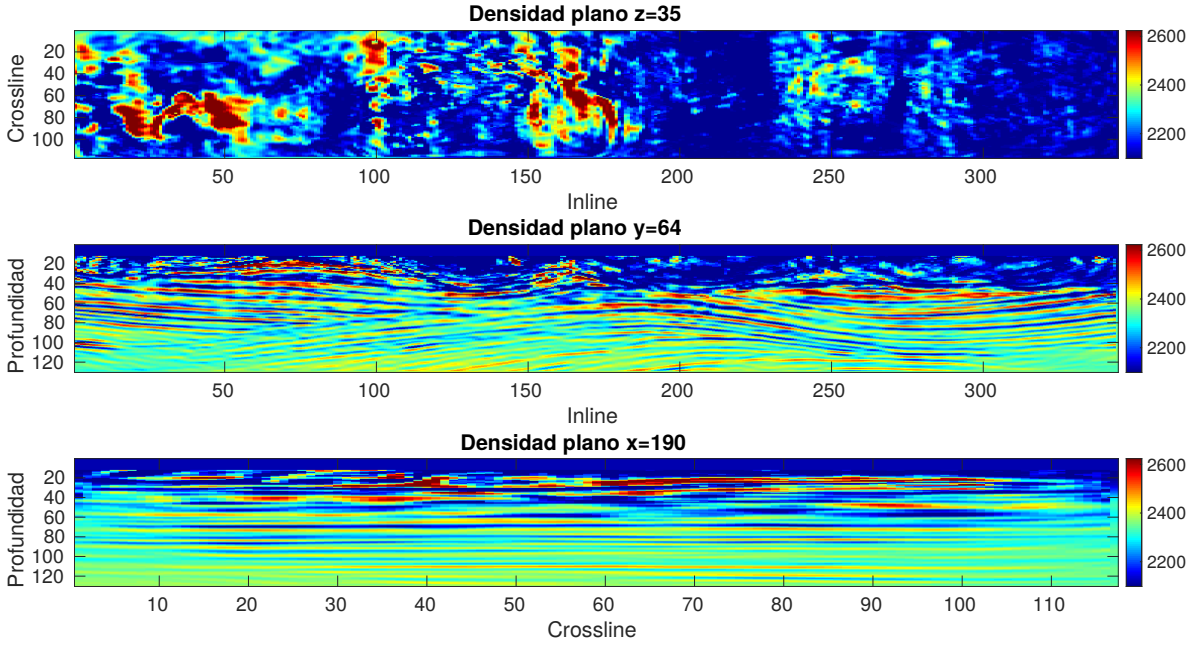


Figura 45. Tres cortes del modelo final de densidades $SEAM\ 3D$.



Nótese que aunque la condición de estabilidad se cumple para este experimento y la cantidad de puntos por longitud de onda es de 7.44, la velocidad y la densidad tuvieron una convergencia no deseada[67] durante el proceso de inversión. Si se compara el comportamiento de los modelos $Overthrust$ y $SEAM$, se puede apreciar que en cualquiera de las tres dimensiones espaciales, la velocidad y densidad para el caso del $SEAM$ tiene mayor complejidad. Esto es, que el modelo $SEAM$ tiene mayor cantidad de variaciones de velocidad y densidad lo cuál puede implicar que se deba usar una fuente con una frecuencia mucho mayor y una resolución espacial más fina que las usadas con el modelo $Overthrust$.

Aunque la medida de PSNR indica un comportamiento general (y nula para el caso de procesamiento de datos reales) se obtuvo un valor de $28.1394[dB]$ para el modelo inicial de velocidades del $Overthrust$ y se incrementó a $30.7134[dB]$ con el modelo final. Para el caso del modelo $SEAM$ el modelo inicial genera un valor de $25.1298[dB]$ y al final del proceso de inversión se tiene un valor de $26.05[dB]$. Si se desea mejorar el resultado se podría realizar un proceso multiescala (como se mencionó al principio de este documento) variando la frecuencia central de la ondícula usada como fuente para encontar con cada paso más detalles en la imagen final pero, como se mencionó previamente, esto implica un sistema de cómputo más robusto del que se disponía para realizar los experimentos.

6.2.3. Porcentaje de utilización de la GPU Mediante la herramienta $nvidia - smi$, se extrajo un documento de texto con la información de utilización de la GPU (en porcentaje) durante la ejecución del método de inversión. Se realizó un promedio de estos valores, ubicados en un rango de 0 a 100, y se obtuvo que durante el proceso de inversión, se tiene una utilización promedio de $95\,\%$ para todas las

[67] Visualmente los modelos deberian tener algunas estructuras similares a los modelos originales pero en el resultado aparecen artefactos que no son coherentes con lo que se espera.

GPUs del cluster. Esto indica que no se está desperdiciando tiempo de ejecución en procesos que no explotan el paralelismo de estos dispositivos o en transferencias de datos (previamente referidos como procesos que implican una gran cantidad de tiempo principalmente si es hacia disco duro mecánico).

En un trabajo de investigación previo [68] se procesaron modelos de velocidades de hasta $250 \times 250 \times 135$ generando campos de propagación de 2000 muestras de tiempo. El proyecto se desarrolló sobre el mismo clúster de cómputo, usaba la ecuación de onda acústica con densidad constante y la ejecución de la implementación tardó alrededor de tres días. Con la estrategia descrita en este documento se puede procesar,en el mismo tiempo de ejecución, alrededor de ocho veces la cantidad de puntos que se logró en el proyecto de referencia.

---

[68] David L Abreo C. "Evaluación de estrategias de implementación de una inversión de onda completa (FWI) 3D acústica con densidad constante en el dominio temporal sobre una arquitectura GPU". Tesis de mtría. XXX: Escuela de ingenierías eléctrica, electrónica y de telecomunicaciones, Universidad Industrial de Santander, 2017.

7. CONCLUSIONES Y TRABAJO FUTURO

EL modelado de la ecuación de onda (acústica, densidad constante o variable, elástica, etc.) es el núcleo de operación del método FWI. Las dimensiones de los modelos de velocidad y densidad (para el caso de este proyecto), resolución espacial, resolución temporal y tiempo de adquisición indican cuántos recursos de hardware serán utilizados en cualquier sistema de cómputo. Es por esto, que es necesario encontrar un balance de utilización del sistema dependiendo de los dispositivos con los que se tenga interés en trabajar.

Para el presente proyecto se definió e implementó una estrategia computacional enfocada en el manejo de recursos de hardware al interior de un sistema de cómputo que dispone tanto de $CPUs$ como de $GPUs$. Esta estrategia se basó en usar la gran capacidad de memoria a la que puede acceder la CPU y la capacidad de cómputo en paralelo de las $GPUs$. Con la estrategia implementada, teniendo en cuenta las carácteristicas de $hardware$ del clúster CPS descrito en la sección tres, se pueden procesar modelos de velocidad y densidad de máximo 512x512x140 puntos, usando una fuente (o un hipercubo de campo de presión) de 5000 muestras de tiempo. La estrategia se definió para usarse en cualquier sistema y la única limitante es qué tan robusto es el equipo de cómputo sobre el que se trabaja. Si se desea procesar un modelo de dimensiones más grandes en el sistema actual, por ejemplo, se debe incrementar la cantidad de RAM disponible en cada nodo de cómputo.

Se logró definir una estrategia versátil que permite lidiar modelos de dimensiones masivas, la calidad de los resultados no se ve afectada debido a que no se hacen modificaciones al algoritmo de inversión pero si se aprovecha una arquitectura presente en la mayoría de los clusters de cómputo a nivel mundial.

Se utilizaron dos modelos de velocidad sintéticos ($Overthrust\ 3D$ y $SEAM\ 3D$) para comprobar el funcionamiento de la implementación realizada. Adicionalmente, se mostró mediante la medida de $PSNR$ la calidad de las imágenes obtenidas. Es necesario tener en cuenta que esta métrica funciona cuando se trabaja con datos sintéticos y que solo brinda información general de la comparación de las amplitudes de los elementos que componen los modelos analizados. Para el caso de procesamiento de datos reales se podrían utilizar medidas como el porcentaje de $cycle\ skipping$ (usada en el trabajo de [69]) o correlación (usada en el trabajo de [6]).

La calidad de las imágenes finales del proceso de inversión puede mejorarse realizando un proceso multi-escala en donde se varía la frecuencia central de la ondícula utilizada como fuente. Sin embargo, dadas las características de hardware del clúster de cómputo $Tokyo$, incrementar la frecuencia implica generar modelos de dimensiones mayores a las máximas que se calcularon con las relaciones 33 y 34. Esto signifa que es necesaria una actualización de hardware con $GPUs$ que dispongan de mayor cantidad de RAM (e.g. La $GPU\ NVIDIA\ TESLA\ V100$ dispone de $32[GB]$ de memoria RAM) y mayor capacidad de memoria de CPU con el fin de continuar el proceso de inversión para frecuencias más altas.

Debido a que el presente trabajo de investigación tuvo un enfoque al manejo de recursos de cómputo, fue necesario medir qué tan eficiente es la estrategia implementada. Por un lado, aunque el tiempo de ejecución indica un desempeño general pero que puede ser subjetivo, también se calculó la cantida de $FLOPs$ de cada

---

[69] Jheyston O Serrano y col. “A cycle-skipping analysis in transformed domains for full waveform inversion using particle swarm optimization (PSO)”. En: *15th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 31 July-3 August 2017*. Brazilian Geophysical Society. 2017, págs. 1787-1792.

una de las funciones que conforman el modelado de la ecuación de onda (la etapa que mayor trabajo realiza dentro del método de inversión). La medida de $FLOPs$ se puede ver como un porcentaje dependiendo de los procesadores que se estén utilizando y este valor deben mantenerse cuando se ejecutan en diferentes equipos de cómputo. Para el caso de este trabajo, se incrementó el rendimiento en $FLOPS$ desde $1.39\,\%$ hasta $3\,\%$.

Por otro lado, el uso de memoria es una medida muy importante debido a que los objetivos de este proyecto estaban enfocados al procesamiento de datos de dimensiones masivas. Es por esto que se debe tener en cuenta tanto el uso de RAM de la GPU como el uso de RAM de la CPU. Para el caso de la GPU se aprovecharon las ventajas brindadas por la mayoria de los tipos de memoria disponibles en el dispositivo. Adicionalmente, la naturaleza del algoritmo de inversión implica que la medida de ancho de banda de memoria es fundamental para conocer el rendimiento de la estrategia implementada. Se logró un incremento del $28\,\%$ al $70\,\%$, aproximadamente, del valor máximo de ancho de banda de la GPU con la implementación desarrollada.

La metodología de implementación de FWI 3D mostrada en este trabajo es independiente de la ecuación de onda utilizada. Gracias a esto se pueden utilizar operadores más complejos como la ecuación de onda con anisotropía y la ecuación de onda elástica. Sin embargo, si se cambia el operador usado se deben re-definir las expresiones de uso de memoria debido a que éstos operadores utilizan volúmenes de datos adicionales. Para el caso de la ecuación que estima el tiempo de ejecución se deja la misma expresión.

Aunque el uso de memoria compartida no fue de utilidad en el diseño e implementación de la estrategia mostrada en este trabajo, se analizaron y utilizaron otras carácteristicas de la GPU para mejorar el rendimiento de un $kernel$ de $CUDA$. Esto brinda otro punto de vista para que futuros desarrollos dentro del área (computación de alto desempeño) no estén limitados a un solo tipo de memoria o un tipo de recurso en específico.

Es necesario realizar procesos adicionales para mejorar el modelo final de densidades. Aunque existen trabajos que relacionan su baja calidad final con el patrón de radiación [70] o un efecto de interferencia conocido como ruido $cross - talk$ [71] es un tema que puede ser investigado en mayor profundidad.

Como trabajo adicional, se propone analizar con más detalle el efecto de la dispersión numérica sobre los modelos finales de velocidad y densidad. Esto con el fin de obtener imágenes de mejor calidad sin tener impacto negativo sobre la estrategia implementada y el rendimiento de la misma.

Para el caso del hardware disponible se recomienda actualizar el sistema de cómputo (tanto $GPUs$ como cantidad de RAM de CPU) si se desean procesar modelos de mayores dimensiones a las que se permiten con las relaciones 33 y 34.

Por último, se propone revisar características como la memoria unificada ($Unified\ memory$ en inglés) que permite dejar la administración de copias de memoria a los

---

[70] Jean Virieux y Stéphane Operto. “An overview of full-waveform inversion in exploration geophysics”. En: *Geophysics* 74.6 (2009), WCC1-WCC26.

[71] Gerard T Schuster y col. “Theory of multisource crosstalk reduction by phase-encoded statics”. En: *Geophysical Journal International* 184.3 (2011), págs. 1289-1303.

controladores de hardware de la GPU o la utilidad de los $Tensor\ cores$ (disponibles en las $GPUs\ TESLA\ V100$). Aunque los $Tensor\ cores$ estan diseñados para aplicaciones de redes neuronales podría explorarse una forma de uso en trabajos futuros así como se aprovecharon los beneficios de la memoria de texturas en el presente trabajo.

BIBLIOGRAFÍA

http://www.sercel.com/about/Pages/what-is-geophysics.aspx. Accedido por última vez: 2018-11-04 (vid. pág. 20).

https://www.stanford.edu. Accedido por última vez: 2018-11-28 (vid. pág. 21).

https://wiki.seg.org/wiki/. Accedido por última vez: 2018-11-28 (vid. pág. 29).

https://www.top500.org/lists/2017/11/. Accedido por última vez: 2018-11-28 (vid. pág. 32).

https://computing.llnl.gov/tutorials/pthreads/. Accedido por última vez: 2018-11-28 (vid. pág. 33).

https://www.openmp.org/. Accedido por última vez: 2018-11-28 (vid. pág. 33).

https://www.open-mpi.org/. Accedido por última vez: 2018-11-28 (vid. pág. 34).

https://developer.nvidia.com/cuda-zone. Accedido por última vez: 2018-11-28 (vid. pág. 34).

https://www.khronos.org/opencl. Accedido por última vez: 2018-11-28 (vid. pág. 34).

https://www.openacc.org/. Accedido por última vez: 2018-11-28 (vid. pág. 34).

https://www.infinibandta.org/. Accedido por última vez: 2018-11-28 (vid. pág. 37).

https://www.linuxjournal.com/article/8368. Accedido por última vez: 2018-11-28 (vid. págs. 39, 58).

http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Accedido por última vez: 2018-11-28 (vid. pág. 42).

https://www.nvidia.com/en-us/data-center/kepler-gpu-architecture/. Accedido por última vez: 2018-11-28 (vid. pág. 44).

https://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html. Accedido por última vez: 2018-11-28 (vid. pág. 46).

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html. Accedido por última vez: 2018-11-28 (vid. pág. 48).

https://www.nvidia.com/content/PDF/fermi_white_papers/. Accedido por última vez: 2018-11-28 (vid. pág. 62).

Abreo, David L, Sergio A Abreo y Ana B Ramirez. “A hybrid methodology for a 3D Full Waveform Inversion in time domain using GPUs”. En: *15th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 31 July-3 August 2017*. Brazilian Geophysical Society. 2017, págs. 369-373 (vid. págs. 48, 53, 54, 66).

Abreo, Sergio y col. “Diving-wave FWI methodology applied to a Colombian Caribbean data set”. En: *The Leading Edge* 37.4 (2018), págs. 291-295 (vid. págs. 21, 95).

Abreo C, David L. “Evaluación de estrategias de implementación de una inversión de onda completa (FWI) 3D acústica con densidad constante en el dominio temporal sobre una arquitectura GPU”. Tesis de mtría. XXX: Escuela de ingenierías eléctrica, electrónica y de telecomunicaciones, Universidad Industrial de Santander, 2017 (vid. pág. 93).

Agudo, Oscar Calderon y col. “3D imaging of the breast using full-waveform inversion”. En: *Proceedings of the International Workshop on Medical Ultrasound Tomography: 1.-3. Nov. 2017, Speyer, Germany*. KIT Scientific Publishing. 2018, pág. 99 (vid. pág. 20).

Amestoy, Patrick y col. “Efficient 3D frequency-domain full-waveform inversion of ocean-bottom cable data with sparse block low-rank direct solver: a real data case study from the North Sea”. En: *SEG Technical Program Expanded Abstracts 2015*. Society of Exploration Geophysicists, 2015, págs. 1303-1308 (vid. pág. 56).

Brocher, Thomas M. “Empirical relations between elastic wavespeeds and density in the Earth’s crust”. En: *Bulletin of the seismological Society of America* 95.6 (2005), págs. 2081-2092 (vid. pág. 82).

Drepper, Ulrich. “What every programmer should know about memory”. En: *Red Hat, Inc* 11 (2007), pág. 2007 (vid. pág. 48).

Ecker, C y col. “An AVO analysis project”. En: () (vid. pág. 21).

Gonzales, Herling. “Taller de Modelado de Propagación de Ondas en Aproximación Escalar y Vectorial Basado en Malla Intercalda (Staggered Grid)”. 2016 (vid. pág. 51).

Hoshino, Tetsuya y col. “CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application”. En: *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE. 2013, págs. 136-143 (vid. pág. 40).

Imbert, David y col. “Tips and tricks for Finite difference and i/o-less FWI”. En: *SEG Technical Program Expanded Abstracts 2011*. Society of Exploration Geophysicists, 2011, págs. 3174-3178 (vid. pág. 56).

Lecomte, JC, E Campbell y J Letouzey. “Building the SEG/EAEG overthrust velocity macro model”. En: *EAEG/SEG Summer Workshop-Construction of 3-D Macro Velocity-Depth Models*. 1994 (vid. pág. 82).

Liu, D. C. y J. Nocedal. “On the Limited Memory BFGS Method for Large Scale Optimization”. En: *Math. Program.* 45.3 (1989), págs. 503-528. DOI: 10.1007/BF01589116 (vid. pág. 29).

Liu, Lu y col. “3D hybrid-domain full waveform inversion on GPU”. En: *Computers & Geosciences* 83 (2015), págs. 27-36 (vid. pág. 57).

Meng, Huan-Ting y Jian-Ming Jin. “Acceleration of the Dual-Field Domain Decomposition Algorithm Using MPI–CUDA on Large-Scale Computing Systems”. En: *IEEE Transactions on Antennas and Propagation* 62.9 (2014), págs. 4706-4715 (vid. pág. 52).

Métivier, Ludovic y col. “Overcoming Cycle Skipping in FWI - An Optimal Transport Approach”. En: (mayo de 2016) (vid. pág. 22).

Nocedal, J. y S. J. Wright. *Numerical Optimization*. 2nd. New York: Springer, 2006 (vid. pág. 27).

Noriega, Reynaldo F. y col. “Implementation strategies of the seismic Full Waveform Inversion”. En: *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE. 2017, págs. 1567-1571 (vid. págs. 53, 54, 66).

Schuster, Gerard T y col. “Theory of multisource crosstalk reduction by phase-encoded statics”. En: *Geophysical Journal International* 184.3 (2011), págs. 1289-1303 (vid. pág. 97).

Serrano, Jheyston O y col. “A cycle-skipping analysis in transformed domains for full waveform inversion using particle swarm optimization (PSO)”. En: *15th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 31 July-3 August 2017*. Brazilian Geophysical Society. 2017, págs. 1787-1792 (vid. págs. 26, 95).

Sullivan, Dennis M. *Electromagnetic simulation using the FDTD method*. John Wiley & Sons, 2013 (vid. pág. 23).

Tarantola, A. “Inversion of seismic-reflection data in the acoustic approximation”. En: *Geophysics* 48.8 (1984), págs. 1259-1266 (vid. pág. 26).

Virieux, Jean y Raul Madariaga. “Dynamic faulting studied by a finite difference method”. En: *Bulletin of the Seismological Society of America* 72.2 (1982), págs. 345-369 (vid. pág. 23).

Virieux, Jean y Stéphane Operto. “An overview of full-waveform inversion in exploration geophysics”. En: *Geophysics* 74.6 (2009), WCC1-WCC26 (vid. pág. 97).

Volkov, Vasily y James W Demmel. “Benchmarking GPUs to tune dense linear algebra”. En: *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*. IEEE. 2008, págs. 1-11 (vid. pág. 81).

Vollmer, Michael y col. “Meta-programming and Auto-tuning in the Search for High Performance GPU Code”. En: *Proceedings of the 4th ACM SIGPLAN Workshop on Functional High-Performance Computing*. ACM. 2015, págs. 1-11 (vid. pág. 62).

Wang, XM. “Seismic wave simulation in anisotropic media with heterogeneity using a high-order finite-difference method”. En: (2001) (vid. pág. 51).

Wesdorp, Jaap J. *A general recipe for obtaining adjoint equations and gradients and its application to full waveform inversion of 2D isotropic elastic media in finite difference time-domain*. Inf. téc. Master internship report. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones, Universidad Industrial de Santander, 2018 (vid. pág. 27).

ANEXOS

Anexo A. IMPLEMENTACIÓN FWI 3D SOBRE EL CLUSTER TOKYO

7.1. Requerimientos de sistema

Esta herramienta fue desarrollada usando un cluster compuesto por tres nodos con las siguientes características (por nodo):

- CPU: Intel Xeon E5-2670 v3 / Intel Xeon CPU E5-2620 v3
- GPU: Nividia Tesla K40c / Nividia Tesla K40m
- RAM: 192 GB DDR4
- S.O.: Debian 8-Jessie / Linux 3.16.0-4-amd64

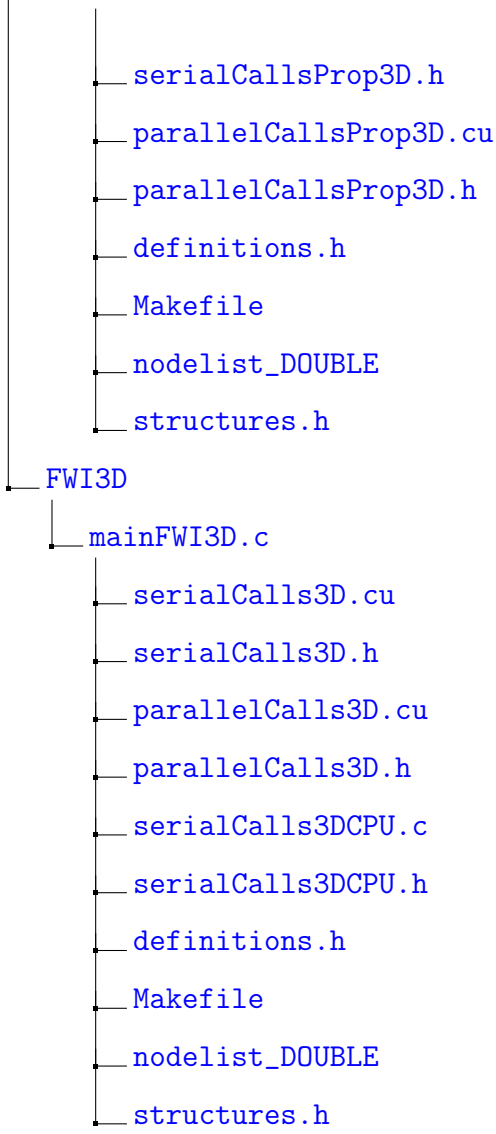
De manera general se recomienda un equipo de cómputo con:

- S.O.: Linux Debian o Ubuntu
- CPU: Cualquier procesador intel certificado en el sitio web oficial `https://ark.intel.com/`
- GPU: NVIDIA TESLA o NVIDIA GeForce con arquitectura Kepler o superior según los sitios web `https://www.nvidia.com/en-us/data-center/tesla/` y `https://www.nvidia.com/en-us/geforce/`
- RAM: La cantidad de RAM disponible en el sistema limita el tamaño de los modelos y la cantidad de muestras de la fuente que se desean utilizar. Es necesario revisar las ecuaciones 33 y 34 para obtener las dimensiones máximas que se pueden procesar con el sistema de cómputo disponible.

7.2. Uso de la herramienta

7.2.1. Organización de archivos Es importante entender la organización de los archivos que conformar este proyecto antes de intentar utilizarlo. En el siguiente árbol de carpetas, se puede apreciar cómo se encuentran alamcenados los códigos fuentes para los dos proyectos que se documentan en este texto. Proyecto PROGATOR3D que se usa para el modelado del campo de presión y obtener los datos modelados. proyecto FWI3D que se usa para realizar el proceso de inversión de onda completa. Para el caso de PROPAGATOR3D solo se debe prestar atención a la carpeta PROPAGATOR3D_v2 que es donde se encuentran la última versión de la función de propagación. Estos proyectos han sido mantenidos bajo Git y GitLab y el último commit de cada uno es el que se ha utilizado para la prueba de datos reales reportada en el último informe de OPS del año 2018 por parte del autor de este documento.

```
Carpeta principal
└── PROPAGATOR3D
    ├── PROPAGATOR3D_v1
    │   ├── mainPropagator3D.c
    │   ├── serialCallsProp3D.cu
    │   ├── serialCallsProp3D.h
    │   ├── parallelCallsProp3D.cu
    │   ├── parallelCallsProp3D.h
    │   ├── definitions.h
    │   ├── Makefile
    │   └── nodelist_DOUBLE
    └── PROPAGATOR3D_v2
        ├── mainPropagator3D.c
        └── serialCallsProp3D.cu
```



Los archivos de extensión `.cu` contienen funciones que se ejecutan en la GPU (kernels) o funciones que configuran y ejecutan kenels sobre la GPU. Los archivos cuyo nombre inicia en $main$ y de extensión `.c` son los archivos principales donde se configura los parámetros o características de cada experimento, entradas y salidas. Los archivos con extensión `.h` pueden ser prototipos de función, definición de estructuras o definiciones de variables globales para cada proyecto. Adicionalmente, para el proyecto de FWI3D existe un archivo `.c` cuyo nombre finaliza en CPU. Este archivo contiene las rutinas que se ejecutan únicamente en CPU y que además hacen

uso del API de OpenMP para aprovechar los múltiples hilos de los procesadores en cada nodo del cluster. El makefile se utiliza para la compilación y ejecución de cada proyecto y el archivo `nodelist_DOUBLE` es usado para definir los nombres de los hosts del cluster sobre el que se va a ejecutar el proyecto (tema relacionado con el estándar de programación paralela MPI - Message Passing Interface).

7.2.2. Archivos clave Antes de empezar a usar la herramienta es importante enteder cuáles son los archivos que podrían ser modificados para cambiar de un experimento a otro y los archivos de entrada y salida (formato y organización)

Codigo fuente Considerando que el lector de este documento es solamente usuario y tiene conocimientos básicos de programación en C sólo será necesario modificar los archivos `definitions.h`, `mainPropagator3D.h` y `mainFWI3D.h`.

- `definitions.h`: Es un archivo de definiciones mediante la instrucción `#define`. Las únicas variables a modificar son `WATER_LAYERS` que se usa para determinar la cantidad de capas de agua en puntos o pixeles que tiene el modelo y `L_BFGS_LAYERS` que se usa para definir la cantidad de "históricos" en la función de $L\text{-}BFGS$. Para el caso de `WATER_LAYERS` se recomienda usar un valor igual o superior a 8 teniendo en cuenta que el modelo a procesar se debe ajustar de la misma forma.

- `mainPropagator3D.c`: Es el archivo principal que realiza la simulación del campo hacia adelante para extraer el dato modelado. Para el caso de datos sintéticos se usa para simular el dato observado que será entrada al archivo `mainFWI3D.c`. La inicialización de parámetros se puede ver en la lista 7.1 y la carga de binarios es como aparece en la lista 7.2. Para el caso de inicialización de parámetros, la tabla 9 resume los más importantes para cambiar las características de cada experimento.

- `mainFWI3D.c`: Es el archivo principal que realiza el proceso de FWI 3D. La inicialización de parámetros de este archivo aparece en la lista 7.3 y las instrucciones de carga de archivos aparecen en la lista 7.4. Para el caso de inicialización de parámetros, la tabla 10 resume los más importantes para cambiar las características de cada experimento.

Listing 7.1. Porción de código del archivo mainPropagator3D.c donde aparecen la inicialización de parámetros.

```
...
//Parameter setup of forward modeling
  paramSetup->propCols = 768;
  paramSetup->propRows = 120;
  paramSetup->propLayers = 256;
  paramSetup->deltaT = 1e-3;
  paramSetup->propTime = 3.50;
  paramSetup->timeSamples =
  round(paramSetup->propTime/paramSetup->deltaT);
  paramSetup->deltaX = 25.0/2;
  paramSetup->deltaY = 25.0/2;
  paramSetup->deltaZ = 25.0/2;
  paramSetup->wCPML = 20;
  paramSetup->R = 1e-4;
  paramSetup->frequency = 3.0;
  paramSetup->numberSources = 100;
  paramSetup->numberRecsX = 480;
  paramSetup->numberRecsY = 6;
  paramSetup->recsDepth = 4;
  paramSetup->sourceDistanceX;
  paramSetup->sourceDistanceY;
```

```
  paramSetup->sourceDepth;
...
```

Tabla 9. Parámetros de inicialización del módulo de propagación. De acuerdo a los requerimientos de cada prueba ejecutada, se debe modificar cada variable manualmente.

Variable	Descripción
propCols	Cantidad de puntos en el eje X - Inline
propRows	Cantidad de puntos en el eje Y - Crossline
propLayers	Cantidad de puntos en el eje Z - Profundidad
detalX	Paso espacial, en metros, en el eje X
detalY	Paso espacial, en metros, en el eje Y
detalZ	Paso espacial, en metros, en el eje Z
detalT	Paso temporal, en milisegundos
propTime	Tiempo de simulación del campo de presión, en segundos
WCPML	Ancho en pixeles de la frontera CPML
frequency	Frecuencia central de la fuente
numberSources	Cantidad de fuentes
numberRecsX	Cantidad de receptores, para cada streamer de cada fuente
numberRecsY	Cantidad de streamers, para cada fuente
sourceDistanceX	Posición Inline de cada fuente
sourceDistanceY	Posición Crossline de cada fuente
sourceDepth	Profundidad de cada fuente

Listing 7.2. Porción de código del archivo mainPropagator3D.c donde aparecen las instrucciones para los binarios de entrada.

```
...
//Loading binary files
  FILE *fid;
  fid=fopen("path-to-velocity-model.bin","rb");
  fread(param->hostVelocity,paramSetup->cubeSize,
  sizeof(FLOAT),fid);
  fclose(fid);

  fid=fopen("path-to-density-model.bin","rb");
  fread(param->hostDensity,paramSetup->cubeSize,
  sizeof(FLOAT),fid);
  fclose(fid);

  fid=fopen("path-to-source-file.bin","rb");
  fread(srcNRec->hostSource,paramSetup->timeSamples,
  sizeof(FLOAT),fid);
  fclose(fid);

  fid=fopen("path-to-receivers-Xcoor.bin","rb");
  fread(srcNRec->hostRecsX,
  paramSetup->numberRecsX*paramSetup->numberRecsY*
  paramSetup->numberSources,sizeof(INTEGER),fid);
  fclose(fid);

  fid=fopen("path-to-receivers-Ycoor.bin","rb");
  fread(srcNRec->hostRecsY,
  paramSetup->numberRecsX*paramSetup->numberRecsY*
```

```
  paramSetup->numberSources,sizeof(INTEGER),fid);
  fclose(fid);

  fid=fopen("path-to-source-coordinates.bin","rb");
  fread(srcNRec->sourcePos,3*
  paramSetup->numberSources,sizeof(INTEGER),fid);
  fclose(fid);
...
```

Listing 7.3. Porción de código del archivo mainFWI3D.c donde aparece la inicialización de parámetros.

```
...
  //Parameter set-up
  paramSetup->propCols = 768;
  paramSetup->propRows = 120;
  paramSetup->propLayers = 256;
  paramSetup->deltaT = 1e-3;
  paramSetup->propTime = 3.50;
  paramSetup->timeSamples =
  round(paramSetup->propTime/paramSetup->deltaT);
  paramSetup->deltaX = 25.0/2;
  paramSetup->deltaY = 25.0/2;
  paramSetup->deltaZ = 25.0/2;
  paramSetup->wCPML = 20;
  paramSetup->R = 1e-4;
  paramSetup->frequency = 7.0;
  paramSetup->numberSources = 100;
  paramSetup->numberRecsX = 480;
  paramSetup->numberRecsY = 6;
  paramSetup->recsDepth = 4;
  paramSetup->sourceDistanceX;
  paramSetup->sourceDistanceY;
  paramSetup->sourceDepth;
  paramSetup->cubeSize=paramSetup->propCols*
  paramSetup->propRows*paramSetup->propLayers;
  FLOAT alpha=10.0;
  FLOAT beta=10.0;
  INTEGER fwiIter=20;
```

```
  INTEGER iteration=0;
  FLOAT *phi = (FLOAT *)calloc(fwiIter,sizeof(FLOAT));
  FLOAT phiAccum=0.0;
  FLOAT normVel, normDen;
  INTEGER srcIter=0;
  FLOAT measuredTime,iniTime,endTime;
  INTEGER counter=0;
  INTEGER *recsYCoor = (INTEGER *)calloc(2*paramSetup->numberRecsY,
  sizeof(INTEGER));
...
```

Tabla 10. Párametros de inicialización del módulo de FWI. De acuerdo a los requerimientos de cada prueba ejecutada, se debe modificar cada variable manualmente.

Variable	Descripción
propCols	Cantidad de puntos en el eje X - Inline
propRows	Cantidad de puntos en el eje Y - Crossline
propLayers	Cantidad de puntos en el eje Z - Profundidad
detalX	Paso espacial, en metros, en el eje X
detalY	Paso espacial, en metros, en el eje Y
detalZ	Paso espacial, en metros, en el eje Z
detalT	Paso temporal, en milisegundos
propTime	Tiempo de simulación del campo de presión, en segundos
WCPML	Ancho en pixeles de la frontera CPML
frequency	Frecuencia central de la fuente
numberSources	Cantidad de fuentes
numberRecsX	Cantidad de receptores, para cada streamer de cada fuente
numberRecsY	Cantidad de streamers, para cada fuente
sourceDistanceX	Posición Inline de cada fuente
sourceDistanceY	Posición Crossline de cada fuente
sourceDepth	Profundidad de cada fuente
alpha	Paso de avance de velocidad en gradiente descendente
beta	Paso de avance de densidad en gradiente descendente
fwiIter	Cantidad de iteraciones de FWI

Listing 7.4. Porción de código del archivo mainPropagator3D.c donde aparecen las instrucciones para los binarios de entrada.

```
...
  //Loading binary files
  FILE *fid, *fid2, *fid3;

    fid=fopen("path-to-velocity-model.bin","rb");
  fread(param->hostVelocity,paramSetup->cubeSize,
  sizeof(FLOAT),fid);
  fclose(fid);

  fid=fopen("path-to-density-model.bin","rb");
  fread(param->hostDensity,paramSetup->cubeSize,
  sizeof(FLOAT),fid);
  fclose(fid);

  fid=fopen("path-to-source.bin","rb");
  fread(srcNRec->hostSource,paramSetup->timeSamples,
  sizeof(FLOAT),fid);
  fclose(fid);

  fid=fopen("path-to-receivers-Xcoor.bin","rb");
  fread(srcNRec->hostRecsX,
  paramSetup->numberRecsX*paramSetup->numberRecsY*
  paramSetup->numberSources,sizeof(INTEGER),fid);
  fclose(fid);

  fid=fopen("path-to-receivers-Ycoor.bin","rb");
  fread(srcNRec->hostRecsY,
```

```
  paramSetup->numberRecsX*paramSetup->numberRecsY*
  paramSetup->numberSources,sizeof(INTEGER),fid);
  fclose(fid);

  fid=fopen("path-to-source-coordinates.bin","rb");
  fread(srcNRec->sourcePos,3*
  paramSetup->numberSources,sizeof(INTEGER),fid);
  fclose(fid);
...
```

Adicionalmente, la inicialización de parámetros se crea al interior de los archivos principales `mainPropagator3D.c` y `mainFWI3D.c` un vector que contiene la cantidad de fuentes que se van a procesar por cada nodo dentro de un clúster de cómputo. En la lista 7.5 se aprecia como se crea este arreglo para el caso de un sistema de cómputo con seis nodos (arraglo de seis elementos) y donde el primer nodo procesará dos fuentes y los demás nodos procesarán una fuente cada uno. Se debe cumplir que la suma de los valores asignados a cada posición de este arrelo sea igual al valor asignado a la variable `numberSources` en la inicialización de parámetros de cualquier proyecto (PROPAGATOR3D o FWI3D).

Listing 7.5. Porción de código del archivo mainPropagator3D.c donde aparecen las instrucciones para los binarios de entrada.

```
...
INTEGER nSRank[]={2,1,1,1,1,1};
...
```

Si se desea trabajar sobre un único sistema con un único nodo de cómputo (workstation, equipo de escritorio o equipo portátil) se puede definir este arreglo tal como aparece en la lista 7.6. Para este caso el único nodo de cómputo para trabajar (arre-

glo de un solo elemento) procesará diez fuentes.

Listing 7.6. Porción de código del archivo mainPropagator3D.c donde aparecen las instrucciones para los binarios de entrada.

```
...
INTEGER nSRank[]={10};
...
```

Binarios de entrada y salida Debido a que en memoria los volúmenes o matrices usados durante el proceso de inversión solo se ven como un arreglo unidimensional de datos, es necesario establecer la organización o forma acceso a cada elemento de estos conjuntos (modelo de velocidad, modelo de densidad, datos modelados, etc.). En la figura 19 se puede ver cómo se mapean los elementos de cualquier volumen de datos usado al interior de la implementación a un arreglo en memoria. Con ayuda de la expresión 36 se puede entender mejor cómo se realiza este proceso.

$$idx = x + propCols * y + propCols * propRows * z; \tag{36}$$

Donde x es una variable que toma valores entre 0 y $propCols$, y es una variable que toma valores entre 0 y $propRows$ y z es una variable que toma valores entre 0 y $propLayers$. De esta forma, se accede a cada uno de los elementos de este volumen. Para el caso de los datos modelados, observados y residuales, x toma valores entre 0 y $numberRecsX$, y‘ toma valores entre 0 y $numberRecsY$ y z toma valores entre 0 y $timeSamples$.

Así como los volúmenes generados al interior de cada proyecto (PROPAGATOR3D y FWI3D), las entradas, cuyas instrucciones se muestran en las listas 7.2 y 7.4, deben organizarse de la misma forma como se ha mostrado en la figura 19. Los archivos

de entrada son de extensión `.bin` y deben estar en formato de punto flotante de precisión simple o formato entero de precisión simple.

7.2.3. Compilación y ejecución

Datos sintéticos Luego de definir las rutas de los archivos de entrada (velocida, densidad, fuente, coordenadas de fuente y coordenadas de receptores), se deben reemplazar estas direcciones en las respectivas instrucciones de lectura de datos dentro del archivo `mainPropagator3D.c` tal como se mostró en la lista 7.2. Adicionalmente se debe definir la ruta del dato modelado, el cuál es el archivo de salida, tal como se muestra en la lista 7.7.

Listing 7.7. Porción de código del archivo mainPropagator3D.c donde aparecen las instrucciones para guardar el binario de salida que contiene el dato modelado o dato observado cuando se use un modelo sintético.

```
...
fid=fopen("path-to-modeled-data","wb");
  //some stuff...
    fseek(fid,(srcIter+srcRankOffset)*
    paramSetup->numberRecsX*paramSetup->numberRecsY
    *paramSetup->timeSamples*sizeof(FLOAT),SEEK_SET);
    fwrite(srcNRec->hostGather,
    paramSetup->numberRecsX*paramSetup->numberRecsY*
    paramSetup->timeSamples,sizeof(FLOAT),fid);
  //some stuff...
...
```

Para compilar y ejecutar el módulo de propagación 3D se debe abrir un terminal y ubicarse en la carpeta donde está el archivo Makefile para escribir:

```
$clear
$make clean
$make propagation
```

Con estas instrucciones el archivo makefile creará y ejecutará un binario con las instrucciones necesarias para llevar a cabo una propagación según la inicialización de parámetros y los datos de entrada.

Luego de generar el dato observado de manera sintética con el proyecto PROPAGATOR3D, se debe hacer uso del proyecto FWI3D. Teniendo en cuenta las variables de inicialización de parámetros y los binarios de entrada, se deben modificar estos valores para el experimento que se esté planteando. Dentro del archivo `mainFW3D.c` existen dos secciones de código donde se debe ajustar la ruta para leer el dato observado. Esta ruta debe ser la misma donde se ha guardado después de ejecutar el proyecto PROPAGATOR3D.

Para compilar y ejecutar el módulo de FWI 3D se debe abrir un terminal y ubicarse en la carpeta donde está el archivo Makefile para escribir:

```
$clear
$make clean
$make
```

Con estas instrucciones se creará y ejecutará la aplicación que entrega a la salida un modelo final de velocidad y un modelo final de densidad.