Optimización del tiempo de ejecución del algoritmo RTM 3D sobre una plataforma GPU.

William Alexander Salamanca Becerra

Trabajo de Grado para optar al título de Doctor en Ingeniería

Directora

PhD. Flor Alba Vivas Mejía

Doctora en Geofísica

Codirectora

PhD. Ana Beatriz Ramírez Silva

Doctora en Ingeniería Eléctrica .

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2019

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Lista de Apéndices

pág.

Resumen

Título: Optimización del tiempo de ejecución del algoritmo RTM 3D sobre una plataforma GPU. [*]

Autor: William Alexander Salamanca Becerra [**]

Palabras Clave: Migración Sísmica Reversa en Tiempo, Ecuación de onda acústica, Estrategias de manejo de memoria, Unidad de Procesamiento Gráfico (GPU), Modelos de predicción de tiempo de ejecución.

Descripción: La exploración sísmica permite tomar decisiones importantes en la industria del petróleo gracias a la estimación de propiedades y a la información estructural del subsuelo obtenidas mediante procesamiento de datos sísmicos. La migración sísmica reversa en tiempo (RTM) es la técnica que mejores resultados ofrece en la generación de la imagen final; pero así mismo es una de las técnicas más costosas del procesamiento de datos.

Durante el desarrollo de la tesis, se estudiaron aspectos de la implementación del algoritmo RTM como: la ecuación de onda, su solución numérica mediante GPU, la dependencia de datos en RTM, las estrategias de manejo de memoria y la estimación del tiempo de ejecución del algoritmo mediante modelos matemáticos. Éstos últimos fueron empleados para la medición del desempeño de la implementación y la medición de la aceleración después de cada optimización. Los principales aportes de la tesis en el área geofísica se concentraron en el análisis de estrategias de manejo de memoria, principalmente la estrategia de fronteras aleatorias que reduce la cantidad de cómputo y memoria significativamente; mientras que en computación, el aporte fue el desarrollo e implementación de modelos de predicción de tiempos de ejecución para medir objetivamente el rendimiento de la implementación.

---

[*] Tesis Doctoral

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Directora: Flor Alba Vivas, Doctora en Geofísica.

Abstract

Title: Optimization of the execution time of the RTM 3D algorithm over a GPU platform. [*]

Author: William Alexander Salamanca Becerra [**]

Keywords: Reverse Time Migration, Acoustic Wave Equation, Memory Management Strategies, Graphics Processing Units (GPU) , Predictive Model for Execution Time.

Descripción: The seismic exploration allows oil industry to make important decisions thanks to the estimation of properties and the structural information of the subsoil obtained by seismic data processing. Reverse Time Migration (RTM) is the technique that offers the best results in the generation of the final image; but it is also one of the most expensive data processing techniques.

During the development of this thesis, several aspects of the RTM algorithm implementation were studied, such as: the wave equation, its numerical solution using GPU, data dependency in RTM, memory management strategies and the estimation of the execution time of the algorithm using mathematical models. The latter were used to estimate: the performance of the implementation and the acceleration after each optimization. The main contributions of the thesis in the geophysical area were focused on the analysis of memory management strategies, mainly the random boundaries strategy that reduces the amount of computation and memory significantly; while in computer sciences, the contribution was the development and implementation of models for predicting execution time to objectively measure the performance of the implementation.

---

[*] Research work

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Directora: Flor Alba Vivas, Doctora en Geofísica.

Introducción

La principal técnica de exploración de hidrocarburos es el experimento sísmico de reflexión. Esta técnica permite determinar la estructura del subsuelo por medio del procesamiento del dato sísmico, el cual se obtiene al generar ondas mecánicas mediante fuentes impulsivas que se reflejan en el subsuelo. Cómo resultado de este proceso se obtiene: la imagen sísmica, que será utilizada en interpretación estructural; e información de amplitudes y campos de velocidad que permiten obtener otros parámetros indicadores de presencia de hidrocarburos para que equipos expertos tomen decisiones sobre la viabilidad económica de realizar o no una perforación.

Dentro del procesamiento de datos sísmicos, se destaca la migración sísmica que utiliza la teoría ondulatoria para formar la imagen del subsuelo, desplazando los eventos sísmicos registrados en superficie hasta su posición real. Los algoritmos de migración sísmica en profundidad requieren como entrada un modelo bien definido de velocidades de la zona. Con algoritmos de migración más precisos y modelos de velocidades de mejor resolución la imagen del subsuelo será de mejor calidad.

A través del tiempo se han propuesto diferentes técnicas de solución a la ecuación de onda. Las técnicas basadas en aproximaciones de altas frecuencias que son la base de los algoritmos de migración Kirchhoff (Bleistein (1987)) fueron desarrolladas inicialmente por su bajo requerimiento de cómputo lo que permitió obtener imágenes migradas en tiempos cortos. Posteriormente

aparecieron soluciones que usaron aproximaciones de ecuación de onda de sentido único (OWWE por sus siglas en inglés) en el dominio de la frecuencia en algoritmos como *Phase-shift* (Gazdag (1978)), *Phase-shift plus interpolation* (Gazdag and Sguazzero (1984)) y *Split-Step* (Stoffa et al. (1990)). En ambientes de estructuras complejas como las asociadas a cuerpos de sal, donde se presentan fenómenos como ondas prismáticas y *turning waves* en modelos de velocidad con fuertes variaciones laterales, se obtuvo mejor calidad en la imagen sísmica (Yoon et al. (2003)) mediante implementaciones que emplean la solución directa de la ecuación de onda acústica, conocida como *Reverse-Time Migration* (RTM) Baysal et al. (1983); Whitmore (1983).

El principal inconveniente de los algoritmos RTM es su alto costo computacional (Clapp et al. (2010); Zhu and Lines (1998)), el cual retrasó la implementación industrial hasta la primera década del presente siglo. Su alto costo se debe a que se requiere calcular y almacenar el campo de onda de la fuente y de los receptores para todo el tiempo del registro sísmico. Esto implica cómputo intensivo para extrapolar los campos y alta demanda de memoria para almacenar los mismos.

Las plataformas modernas de cómputo de alto desempeño ofrecen alta capacidad de procesamiento distribuido y logran su mejor desempeño minimizando las transferencias. Por las características del algoritmo RTM3D, la plataforma que más se adapta para su ejecución son las GPU porque concentran la mayor capacidad de cómputo de punto flotante en un solo dispositivo. Plataformas como la DXG-2 de Nvidia concentran hasta 2 petaFLOPS distribuidos en 16 GPU Tesla V100, cada una con 32 GB de RAM disponibleNvi (2019). La capacidad de cómputo y el manejo

de operaciones vectoriales de estas plataformas favorecen notablemente la ejecución del algoritmo RTM y satisfacen su demanda de cómputo. Por otro lado la memoria disponible para almacenar los campos de onda aun es insuficiente El principal problema de las GPU ejecutando RTM3D es la memoria disponible, por ello este trabajo es de gran importancia para explotar la capacidad de cómputo de las GPU bajo la restricción de memoria.

Durante el desarrollo del presente trabajo se encontró que los factores que tienen un mayor impacto sobre el costo computacional son: el tamaño de los datos de entrada, el método numérico empleado para resolver la ecuación de onda, la estrategia de manejo de memoria, el método de frontera absorbente, el método de paralelización y los parámetros propios de la plataforma de cómputo. El efecto que los factores mencionados tienen sobre el tiempo de ejecución, se estudió mediante modelos de predicción del mismo. Para ello se realizó una implementación propia desde cero del algoritmo RTM 3D sobre una plataforma de cómputo basada en GPUs debido a que es la plataforma moderna con mayor grado de paralelismo en el manejo de datos de punto flotante. Esta implementación procesa un disparo en una sola GPU, lo que hace que sea escalable a sistemas de múltiples GPU asumiendo la estrategia de paralelismo más intuitiva donde se migran en paralelo tantos disparos como GPUs se dispongan. La versión inicial de esta implementación se denominó *naive* e incorporó estrategias de manejo de memoria que le permitieron ejecutar el algoritmo empleando únicamente la memoria del dispositivo. Adicionalmente cada uno de los *kernels* de la versión *naive* fue mejorado en el desarrollo de esta tesis.

Para la optimización de la implementación *naive* se propuso una metodología basada en los

modelos de predicción del tiempo de ejecución, lo cual permite estudiar el efecto de los factores que afectan el desempeño y evaluar integralmente la aceleración de las mejoras realizadas sobre el código. Esta metodología se aplicó sobre las rutinas más relevantes ejecutadas en la GPU, llamadas *kernels*, donde inicialmente se estudió el efecto de los argumentos de entrada y finalmente se estudió el impacto de algunos parámetros de hardware que modifican la ejecución del algoritmo y de esta forma encontrar las mejores condiciones para la ejecución de RTM 3D.

Dentro de los aportes más importantes hechos mediante el desarrollo de esta tesis, está la implementación de las estrategias para el manejo de la memoria aplicado sobre el caso una GPU. Se hizo énfasis en el error que introducen las estrategias que reconstruyen el campo de la fuente hacia atrás: almacenando las fronteras y con fronteras aleatorias. Esta última tiene un alto potencial para ser aplicado sobre datos grandes siempre que se pueda controlar el error del campo que impacta la frontera. También se encontraron algunos casos de inestabilidad del esquema de diferencias finitas no reportados en la literatura cuando se varían los valores de velocidad de la frontera aleatoria en el tiempo. Por otro lado en el proceso de implementación, se propuso una metodología que incluye los modelos de predicción del tiempo de ejecución antes y después de las mejoras. De esta forma se pueden identificar los factores más influyentes en el desempeño y se puede evaluar integralmente la aceleración obtenida con las mejoras implementadas.

Este documento presenta en el capítulo 2 los principios teóricos de RTM y de las estrategias de manejo de memoria así como algunos detalles de la implementación en la GPU. Se presentan

los resultados experimentales y las medición de los recursos computacionales al migrar un dato con las estrategias para realizar la comparación tanto de los resultados como del desempeño computacional.

En el capítulo 3 se presenta y se desarrolla la metodología propuesta para mejorar el desempeño de cada una de las rutinas ejecutadas sobre la GPU (*kernels*). En ella se establecen las variables que afectan el tiempo de ejecución y se desarrollan los modelos que permiten calcular la aceleración. Finalmente se evalúa la aceleración obtenida en el procesamiento completo de un disparo. Por último, en el capítulo 4 se presentan las conclusiones obtenidas mediante el desarrollo de la tesis y se da paso a una sección de discusión de temas que pueden seguir siendo objeto de estudio. En el anexo 1 se amplía el desarrollo matemático sobre las fronteras absorbentes CPML del artículo Pasalic and McGarry (2010), dando una interpretación propia del mismo. En el anexo 2 se dan algunos detalles de la implementación realizada como su estructura y los parámetros de entrada.

1. Objetivos

Objetivo General

Modelar y optimizar el tiempo de ejecución de la implementación del algoritmo RTM 3D con ecuación de onda acústica y densidad constante empleando diferencias finitas sobre una GPU, teniendo en cuenta: el tamaño de los datos de entrada, las técnicas de manejo de bordes, las especificaciones de la plataforma de cómputo y las estrategias para la reconstrucción del campo de onda.

Objetivos Específicos

Determinar los parámetros que afectan el tiempo de ejecución del algoritmo RTM sobre GPU para uso industrial en sísmica 3D;

Modelar el tiempo de ejecución de la implementación del algoritmo RTM 3D sobre una GPU;

Implementar el algoritmo RTM 3D sobre una plataforma basada en GPUs que permita modificar los parámetros que afectan el tiempo de ejecución;

Validar los resultados del modelo a través de la implementación del algoritmo;

Determinar los parámetros óptimos del modelo desarrollado que minimicen el tiempo de ejecución de la implementación del algoritmo RTM 3D.

2. Estrategias de Implementación RTM

La migración reversa en tiempo (RTM) es un método de migración sísmica que soluciona la ecuación de onda completa en el dominio temporal por lo que requiere la velocidad del subsuelo, y así genera una mejor imagen sísmica en zonas de complejidad estructural. RTM ha sido empleado con diferentes ecuaciones de onda: acústico (Baysal et al., 1983), elástico (Chang and McMechan, 1987), con anisotropía VTI (Alkhalifah, 2000), anisotropía TTI (Zhou et al., 2006) y otras ecuaciones derivadas de la ecuación elastodinámica. Así mismo se han empleado diferentes métodos numéricos en la extrapolación del campo de onda como: diferencias finitas, métodos espectrales (Fornberg, 1987) , elementos finitos. En todos estos casos, resolver el campo de onda tiene un alto costo computacional y solo los avances tecnológicos han permitido la implementación del algoritmo RTM empleando métodos numéricos y ecuaciones de onda mucho más complejos en proyectos de procesamiento de datos sísmicos 3D en profundidad. Durante el desarrollo de esta tesis doctoral se empleó la ecuación de onda acústica con densidad constante para extrapolar el campo mediante diferencias finitas como un primer paso en el análisis de estrategias de manejo de recursos computacionales que se podrán extender a otros casos.

El desarrollo de los sistemas de cómputo y las herramientas software se ha orientado hacia el cálculo paralelo mediante múltiples unidades de cómputo. Las unidades de procesamiento gráfico (GPU) concentran una gran cantidad de núcleos en un solo dispositivo en arquitecturas *single instruction multiple thread* (SIMT) (NVIDIA Corporation, 2013). Esto favorece la ejecución de

algoritmos con alto potencial para ser paralelizados como RTM (Clapp et al., 2010). Por otro lado RTM demanda de una cantidad considerable de memoria para almacenar los campos de onda de la fuente y de los receptores, requeridos para calcular la condición de imagen. Esto fácilmente puede exceder los límites de memoria de la GPU e implicaría el uso de niveles superiores de jerarquía de memoria, lo cual afectaría el desempeño del algoritmo. Al día de hoy, la GPU Quadro RTX 8000 es la GPU con más memoria disponible con 48GB. La GPU para diseñada para procesamiento de datos con más memoria es la Tesla V100 con 32GB.

Debido al fuerte contraste entre el alto potencial de cómputo, la poca memoria disponible en el dispositivo y la baja velocidad de los buses de comunicación, varios autores han propuesto estrategias que reducen la memoria requerida aumentando el cómputo. Algunas de estas estrategias se han implementado en CPU y pueden ser adaptadas para las GPU.

A lo largo del presente capítulo, se presentarán los detalles de la implementación del algoritmo en una única GPU y la implementación de tres estrategias: la primera guarda solo algunos puntos de control del campo de onda de la fuente, la segunda solo almacena las fronteras y los dos últimos pasos de tiempo para reconstruir el resto del campo en dirección inversa y la tercera usa fronteras aleatorias en el borde del modelo para evitar el almacenamiento de las fronteras del campo.

2.1. Ecuación de onda

De la teoría de la elasticidad se deducen el siguiente conjunto de ecuaciones:

$$\varepsilon_{kl} = \frac{1}{2}\left(\frac{\partial u_k}{\partial x_l} + \frac{\partial u_l}{\partial x_k}\right) \tag{1}$$

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} \tag{2}$$

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i - \rho\frac{\partial^2 u_i}{\partial t^2} = 0 \tag{3}$$

donde ε es el tensor de deformación, u son los desplazamientos, σ es el tensor de esfuerzo, f son las fuerzas de cuerpo, ρ es la densidad y C es el tensor de elasticidad. Éste último también puede ser representado por una matriz 6×6 llamada matriz de Voigh, que en el caso isótropo queda

expresado únicamente en términos de dos parámetros λ y μ así:

$$C = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}. \tag{4}$$

Los parámetros λ y μ son llamados parámetros de Lamé. También se puede representar C de forma compacta mediante deltas de Kronecker

$$C_{ijkl} = \lambda\,\delta_{ij}\delta_{kl} + \mu\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right). \tag{5}$$

Retomando la ecuación 2 y sustituyendo C_{ijkl} se tiene

$$\sigma_{ij} = \left[\lambda\,\delta_{ij}\delta_{kl} + \mu\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right)\right]\varepsilon_{kl} \tag{6}$$

Al simplificar empleando la simetría de los parámetros de Lamé, se obtiene

$$\sigma_{ij} = \lambda\,\delta_{ij}\varepsilon_{kk} + 2\mu\,\varepsilon_{ij} \tag{7}$$

Al sustituir 1 en esta última ecuación se tiene

$$\sigma_{ij} = \lambda \delta_{ij} \left(\frac{\partial u_k}{\partial x_k} \right) + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{x_i} \right) \tag{8}$$

Ahora se sustituye σ_{ij} en la ecuación 3

$$\frac{\partial}{\partial x_j} \left[\lambda \delta_{ij} \left(\frac{\partial u_k}{\partial x_k} \right) + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right] + f_i - \rho \frac{\partial^2 u_i}{\partial t^2} = 0 \tag{9}$$

Teniendo en cuenta que los parámetros λ y μ no dependen de las variables espaciales, se puede desarrollar la expresión así:

$$\lambda \delta_{ij} \left(\frac{\partial}{\partial x_j} \frac{\partial u_k}{\partial x_k} \right) + \mu \left(\frac{\partial}{\partial x_j} \frac{\partial u_i}{\partial x_j} + \frac{\partial}{\partial x_j} \frac{\partial u_j}{\partial x_i} \right) + f_i = \rho \frac{\partial^2 u_i}{\partial t^2} \tag{10}$$

Escribiendo esta expresión en forma vectorial, se tiene

$$\lambda \vec{\nabla} \left(\vec{\nabla} \cdot \vec{\boldsymbol{u}} \right) + \mu \nabla^2 \vec{\boldsymbol{u}} + \mu \vec{\nabla} \left(\vec{\nabla} \cdot \vec{\boldsymbol{u}} \right) = \rho \frac{\partial^2 \vec{\boldsymbol{u}}}{\partial t^2} \tag{11}$$

Empleando la identidad vectorial $\nabla^2 \vec{\boldsymbol{A}} = \vec{\nabla} \left(\vec{\nabla} \cdot \vec{\boldsymbol{A}} \right) - \nabla \times \left(\nabla \times \vec{\boldsymbol{A}} \right)$ se obtiene la siguiente expresión

$$\lambda \vec{\nabla} \left(\vec{\nabla} \cdot \vec{\boldsymbol{u}} \right) + \mu \left[\vec{\nabla} \left(\vec{\nabla} \cdot \vec{\boldsymbol{u}} \right) - \nabla \times (\nabla \times \vec{\boldsymbol{u}}) \right] + \mu \vec{\nabla} \left(\vec{\nabla} \cdot \vec{\boldsymbol{u}} \right) = \rho \frac{\partial^2 \vec{\boldsymbol{u}}}{\partial t^2} \tag{12}$$

$$(\lambda + 2\mu)\,\vec{\nabla}\left(\vec{\nabla}\cdot\vec{\boldsymbol{u}}\right) + \mu\left(\nabla\times(\nabla\times\vec{\boldsymbol{u}})\right) = \rho\frac{\partial^2\vec{\boldsymbol{u}}}{\partial t^2} \tag{13}$$

Finalmente aplicando el operador divergencia sobre toda la expresión, definiendo $p = \vec{\nabla}\cdot\vec{\boldsymbol{u}}$ y aplicando la identidad vectorial $\vec{\nabla}\cdot\left(\vec{\nabla}\times\vec{\boldsymbol{A}}\right) = 0$, se tiene

$$(\lambda + 2\mu)\,\vec{\nabla}\cdot\left[\vec{\nabla}\left(\vec{\nabla}\cdot\vec{\boldsymbol{u}}\right)\right] + \mu\,\vec{\nabla}\cdot\left[\left(\nabla\times(\nabla\times\vec{\boldsymbol{u}})\right)\right] = \rho\,\vec{\nabla}\cdot\left[\frac{\partial^2\vec{\boldsymbol{u}}}{\partial t^2}\right] \tag{14}$$

$$(\lambda + 2\mu)\,\vec{\nabla}\cdot\left[\vec{\nabla}(p)\right] = \rho\left[\frac{\partial^2 p}{\partial t^2}\right] \tag{15}$$

$$\nabla^2(p) = \frac{\rho}{\lambda + 2\mu}\frac{\partial^2 p}{\partial t^2} = \frac{1}{v^2}\frac{\partial^2 p}{\partial t^2} \tag{16}$$

El campo escalar p, por su definición, es el campo de presión acústica y por la forma de la ecuación 16, se identifica una ecuación de onda con velocidad de propagación

$$v = \sqrt{\frac{\lambda + 2\mu}{\rho}} \tag{17}$$

2.2. Algoritmo RTM

RTM procesa los datos agrupados por disparos. En cada uno de los disparos se extrapola el campo de onda, el cual se define como un campo escalar en el espacio x, y, z y en el tiempo t. La extrapolación se realiza en el tiempo desde dos perspectivas:

Campo de la fuente $p_s(x,y,z,t)$**:** Se calcula con la información de la fuente (ondícula y posición) y simula el campo de onda del experimento sísmico desde el tiempo $t = 0$ donde se supone un medio sin perturbaciones, hasta el tiempo final de simulación $t = t_{max}$.

Campo de los receptores $p_r(x,y,z,t)$**:** Se calcula basado en los registros de los geófonos (campo en superficie). Por ello se calcula hacia atrás desde el tiempo final de simulación hasta el tiempo inicial.

La imagen final es el mapa de reflectividad que se calcula sumando los aportes de cada disparo. La condición de imagen es la expresión matemática que representa la estrategia mediante la cual se calcula la reflectividad en cada punto del espacio. Naturalmente ésta busca resaltar los puntos donde coinciden los campos de la fuente y de los receptores. Claerbout (1971) fue pionero en el análisis de las condiciones de imagen que en su forma más sencilla es una correlación cruzada:

$$R(x,y,z) = \sum_{\forall shots} \sum_{t=0}^{t_{max}} p_s(x,y,z,t) \times p_r(x,y,z,t). \tag{18}$$

La extrapolación de estos dos campos debe ser solución de la ecuación de onda. Durante el desarrollo del presente trabajo se empleará la ecuación de onda acústica con densidad constante dada ecuación 16.

2.2.1. Solución numérica (Diferencias finitas). El método más común en la solución de la ecuación de onda es el uso de diferencias finitas que sustituye los diferenciales por sumas ponderadas de valores del campo. Al aplicar una aproximación de segundo orden a la derivada

temporal de la ecuación de onda tenemos:

$$\frac{p^{k-1} - 2p^k + p^{k+1}}{\Delta t^2} = v^2 \bigtriangledown^2 p^k + s^k. \tag{19}$$

donde p representa alguno de los campos de onda p_s o p_r, el superíndice p^k representa el campo p evaluado en el tiempo $t = k\Delta t$. Esta expresión permite generar un esquema para calcular el campo en un tiempo posterior (p^{k+1}) en función del valor del campo en los dos tiempos anteriores (p^k y p^{k-1}). Así mismo se puede generar un esquema para retropropagar el campo despejando p^{k-1}.

Aplicar un orden de aproximación superior al diferencial temporal implica más términos en el lado izquierdo de la ecuación 19 lo cual implica una mayor cantidad de memoria requerida para el cálculo.

Los diferenciales espaciales que conforman el laplaciano también son aproximados con diferencias finitas. En este caso el uso de aproximaciones de orden superior no afecta el uso de memoria, solo incrementa los accesos a memoria. Estas aproximaciones de orden superior mejoran la calidad de la solución, reduciendo fenómenos como la dispersión numérica. (Liu et al., 2010)

Los diferenciales espaciales se aproximaron mediante la expresión

$$\frac{\partial^2 p}{\partial x^2} = \sum_{l=-N/2}^{N/2} \frac{C_l p_{i+l}^k}{\Delta x^2} \tag{20}$$

donde C_l representa los coeficientes de la aproximación, N el orden de la aproximación y Δx es el muestreo espacial del campo de onda.

2.2.1.1. Estabilidad. Toda solución numérica debe garantizar que el esquema de la solución sea estable para llegar a una solución correcta. En este caso, la estabilidad del sistema se garantiza restringiendo Δt según Liu et al. (2010)

$$\Delta t \leq \frac{2\,\text{mín}(\Delta x, \Delta y, \Delta z)}{\sqrt{3} V_{max} \sqrt{-C_0 + \sum_{l=1}^{N/2} C_l (-1)^{l+1}}}. \tag{21}$$

2.2.1.2. Dispersión numérica. Otro fenómeno relevante por analizar es la dispersión numérica que hace que la velocidad de propagación de la energía del campo de onda sea diferente según su frecuencia. Liu et al. (2010) establece tres factores que afectan la dispersión numérica:

El ángulo de propagación: La energía sufre menos dispersión numérica en frentes de onda que viajan a 45 grados en un espacio bidimensional.

Orden de aproximación espacial: A mayor orden de aproximación, menor dispersión.

Puntos por longitud de onda ($pplo$): Tener un muestreo espacial denso del campo de onda, reduce la dispersión numérica.

Solo los dos últimos factores pueden ser controlados, por ello, según el orden escogido por el usuario, la implementación realizada, remuestrea el campo de velocidades garantizando un mínimo de $pplo$. El nuevo muestreo espacial está dado por:

$$\Delta x, \Delta y, \Delta z \leq \frac{V_{min}}{pplo\, f} \tag{22}$$

donde f es la máxima frecuencia de la ondícula.

2.2.2. Condiciones de frontera absorbentes. El uso de un espacio de memoria limitado para el cálculo implica que solo una parte del campo de onda puede ser representada. En los límites del espacio representado se generan fronteras artificiales. Según el método numérico, se genera un efecto no deseado en el campo de onda. Por ello se emplean métodos para desvanecer la energía que alcanza el borde del modelo.

Para el presente trabajo, se empleó *Perfectly Matched Layer* (PML) debido a su eficiencia. La idea original fue desarrollada por Berenger (1994, 1996) para modelar la propagación de ondas electromagnéticas. Posteriormente Chew and Liu (1996) adaptaron la técnica para ser usada en el ámbito de las ondas sísmicas. Durante este trabajo se utilizó la adaptación de Pasalic and McGarry (2010) que se aplica directamente sobre la ecuación de onda con segundas derivadas (16). El principio de CPML es aplicar un factor de escala complejo en el dominio de la frecuencia dado

por

$$s_i = 1 + \frac{\sigma_i}{\alpha_i + j\omega}, \tag{23}$$

a los operadores diferenciales de la ecuación de onda acústica (16) en cada una de las direcciones $i \in \{x, y, z\}$. σ_i y α_i son parámetros que ajustan el comportamiento de la frontera absorbente. Los nuevos operadores diferenciales son

$$\frac{\partial}{\partial i} \rightarrow \frac{\partial}{\partial i} + \psi_i \tag{24}$$

$$\frac{\partial^2}{\partial i^2} \rightarrow \frac{\partial^2}{\partial i^2} + \frac{\partial \psi_i}{\partial i} + \zeta_i \tag{25}$$

donde ψ_i y ζ_i son dos operadores que al ser aplicados sobre el campo de onda, generan campos adicionales que se calculan recursivamente en cada paso de tiempo así:

$$\psi_i p^n = a_i \psi_i p^{n-1} + b_i \left(\frac{\partial p}{\partial i} \right)^n \tag{26}$$

$$\zeta_i p^n = a_i \zeta_i p^{n-1} + b_i \left[\left(\frac{\partial^2 p}{\partial i^2} \right)^n + \left(\frac{\partial \psi_i p}{\partial i} \right)^n \right]. \tag{27}$$

En las ecuaciones anteriores, a_i y b_i están dados por:

$$a_i = e^{(\sigma_i+\alpha_i)\Delta t} \tag{28}$$

$$b_i = \frac{\sigma_i}{\sigma_i+\alpha_i}(a_i-1). \tag{29}$$

La deducción de estas expresiones son parte del trabajo de Pasalic and McGarry (2010) y se encuentra ampliada en el Apéndice 1. Sustituyendo los operadores de segundas derivadas de la ecuación (25) en la ecuación de onda acústica (16), se tiene

$$\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}+\frac{\partial^2 p}{\partial z^2}+\frac{\partial \psi_x p}{\partial x}+\frac{\partial \psi_y p}{\partial y}+\frac{\partial \psi_z p}{\partial z}+\zeta_x p+\zeta_y p+\zeta_z p=\frac{1}{v^2}\frac{\partial^2 p}{\partial t^2}. \tag{30}$$

La ecuación 30 es aplicada cerca de los bordes del modelo en cada paso de tiempo mediante los siguientes pasos:

1. Actualizar $\psi_i p^n$ usando $\psi_i p^{n-1}$ y la primera derivada de p^n (Ecuación 26).

2. Actualizar $\zeta_i p^n$ usando $\zeta_i p^{n-1}$, la segunda derivada de p^n y la primera derivada de $\psi_i p^n$, calculada en el paso anterior (Ecuación 27).

3. Usar la ecuación 30 para calcular p^{n+1}.

Los campos $\psi_i p$ y $\zeta_i p$ requieren memoria extra para cada frontera del modelo. La Figura 1 relaciona de forma gráfica las variables de la Ecuación 30 y los pasos enumerados para su evalua-

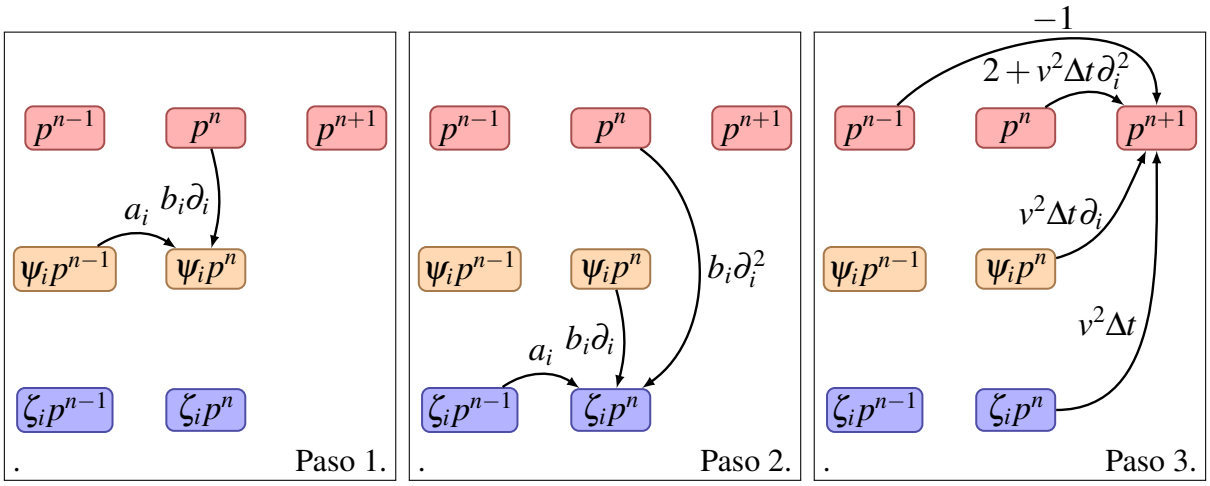


Figura 1. Representación gráfica de las variables que aparecen en la ecuación 30. Su evaluación requiere de tres pasos donde inicialmente se actualiza el campo $\psi_i p^n$ (Paso 1), posteriormente se actualiza el campo $\zeta_i p^n$ (Paso 2) y finalmente se suman los aportes de cada variable para actualizar el campo de onda p^{n+1}(Paso 3).

ción.

2.3. Estrategias de manejo de memoria

La implementación del algoritmo RTM requiere de grandes cantidades de memoria y cómputo, de acuerdo con el tamaño de los datos de entrada. Una implementación ineficiente del algoritmo puede fácilmente exceder los límites de memoria del dispositivo, por lo que es necesario considerar estrategias para reducir el uso de memoria. Estas estrategias liberan parte de la memoria que almacena el campo de onda y luego lo recalculan cuando es requerido. De esta forma, el uso de las estrategias permite que el algoritmo se pueda ejecutar en arquitecturas con alta capacidad de cómputo pero con memoria limitada como las GPU, a cambio de un incremento en la cantidad de operaciones realizadas.

2.3.1. Estimación inicial de memoria. Para procesar un disparo se extrajo la parte del modelo de velocidades que es iluminada por el mismo. Este submodelo puede llegar a ser signi-

ficativamente más pequeño que el modelo completo de velocidades, lo cual representa una gran reducción en los requerimientos de memoria. El submodelo contiene el área del *template*, un área adicional definida por la apertura de migración y un área adicional para la frontera absorbente. La Figura (2) muestra una vista superior de la extracción del submodelo de velocidades y el anexo 2 describe los cálculos realizados en la implementación de este proceso.

A modo de ejemplo, se ha tomado el caso de un submodelo de tamaño $5000 \times 5000 \times 3000$ [m] donde se empleó diferencias finitas de octavo orden en el espacio (`ord=8`) y segundo orden en el tiempo, $L = 8$ capas en la frontera absorbente y $t_{max} = 5s$. El muestreo espacial (`ds`=Δx=Δy=Δz), se ajustó para cumplir un `pplo` $= 10$ según diferentes frecuencias de ondículas como se ve en la Tabla 1. Esto afecta el tamaño del modelo $(\texttt{Nx}, \texttt{Ny}, \texttt{Nz})$, el paso de tiempo `dt` que debe cumplir la condición de estabilidad y los pasos de tiempo de modelado (`itmax`).

Finalmente la Tabla 1 muestra la cantidad de memoria requerida para almacenar el campo en un único paso de tiempo, el campo completo y solo las fronteras del campo para todos los tiempos.

Nótese en la Tabla 1 que para 50 Hz, el sistema de cómputo puede requerir hasta 24 TB de memoria para almacenar el campo de la fuente para todos los 5520 pasos de tiempo. Esta cantidad de memoria podría estar disponible en bajos niveles de jerarquía de memoria, como discos duros, pero usarlos causaría una ejecución muy lenta e ineficiente. Almacenar todo el campo de la fuente no es un método práctico ni siquiera en arquitecturas de CPU donde se dispone de gran cantidad de

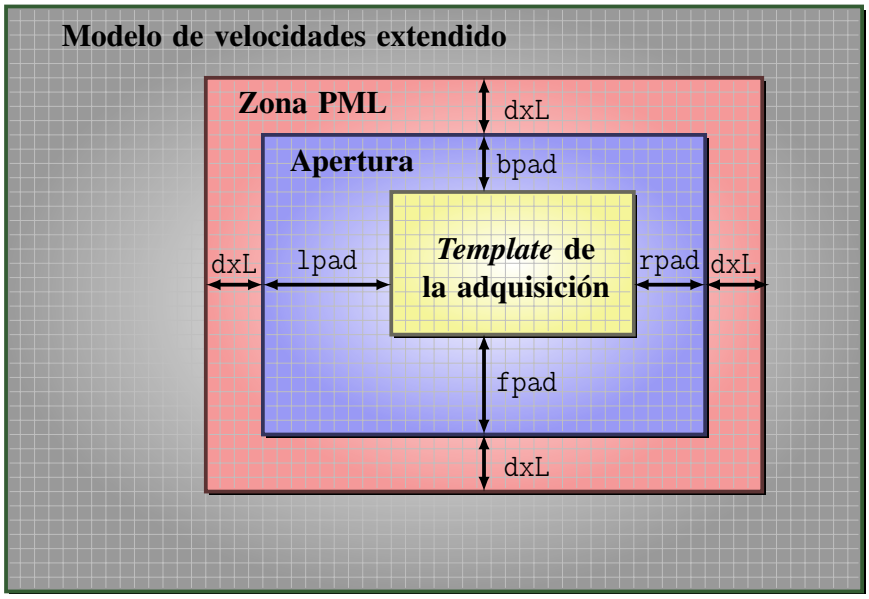


Figura 2. Vista superior del proceso de extracción del submodelo de velocidades. La referencia es el *template* de la adquisición para este disparo. Posteriormente la apertura define un área adicional y finalmente se añade la zona PML.

memoria RAM. En arquitecturas GPU donde la memoria disponible es una limitante, la selección de la estrategia es fundamental para garantizar trabajar solamente en ese nivel de jerarquía de memoria.

2.3.2. Estrategia 1: Puntos de control. Esta estrategia utiliza toda la memoria disponible en el dispositivo de cómputo para almacenar puntos de control durante el proceso de modelado del campo de la fuente. Cuando la condición de imagen requiera un punto no almacenado del campo de la fuente, éste es recalculado desde el punto de control anterior. La distribución de los puntos de control a lo largo de la línea de tiempo fue propuesta inicialmente por Dussaud et al. (2008), quien empleó una distribución uniforme de los puntos de control. Posteriormente Symes (2007) empleó una distribución óptima basada en los algoritmos de Griewank (1992). Finalmente Anderson et al. (2012) ajustó esta optimización debido a que no se tuvo en cuenta la cantidad real de memoria en

Tabla 1
Memoria requerida para almacenar el campo de onda completo para diferentes frecuencias `fq` de la ondícula. El máximo paso en el muestreo espacial(`ds`) lo determina `fq`. Con `ds` se remuestrea el modelo llegando a un tamaño (`Nx,Ny,Nz`). La condición de estabilidad limita `dt` y establece la cantidad de pasos de tiempo del modelado (`itmax`). Los cálculos de los campos de onda se hicieron con representación `float`.

`fq` [Hz]	**ds** [m]	**Nx**	**Ny**	**Nz**	**dt** [ms]	**itmax**	***Snapshot* del campo [MB]**	**Campo completo [GB]**	**Fronteras del campo [GB]**
6	33.33	148	148	88	7.55	662	7.7	5.1	0.7
2	100	50	50	30	22.64	220	0.3	0.1	0.9
10	20	246	246	146	4.53	1104	35.3	39.0	3.8
14	14.29	344	344	204	3.23	1545	96.6	149.2	11.1
18	11.11	442	442	262	2.52	1987	204.7	406.8	24.3
22	9.09	540	540	320	2.06	2429	373.2	906.6	45.3
26	7.69	638	638	378	1.74	2870	615.5	1766.3	75.7
30	6.67	736	736	436	1.51	3312	944.7	3128.9	117.5
34	5.88	834	834	494	1.33	3753	1374.4	5158.2	172.3
38	5.26	932	932	552	1.19	4195	1917.9	8045.7	241.9
42	4.76	1030	1030	610	1.08	4637	2588.6	12003.3	328.2
46	4.35	1128	1128	668	0.98	5078	3399.8	17264.2	432.8
50	4	1226	1226	726	0.91	5520	4364.9	24094.4	557.7

cada punto de control.

Un punto de control contiene los datos necesarios para reanudar el proceso de modelado hacia adelante sin introducir error. Se puede ver por la dinámica de las variables en la Figura 1 que para un punto de control es necesario almacenar únicamente el campo en dos pasos de tiempo consecutivos y un paso de tiempo de las variables auxiliares de CPML.

2.3.3. Estrategia 2: Guardar las fronteras. En la estrategia anterior, el campo de la fuente es recalculado siempre hacia adelante. Sin embargo, el proceso de propagación es reversible al

despejar el término p^{i-1} en la ecuación 19 lo cual permite propagar el campo de la fuente hasta t_{max} y después recalcularlo todo hacia atrás, almacenando únicamente el campo en los dos últimos pasos de tiempo.

El principal problema con esta estrategia es la energía en las fronteras, porque la frontera absorbente disipa la energía que alcanza el borde del modelo. Esto significa que para un paso de tiempo, el campo en el interior del modelo puede ser reconstruido pero la zona donde PML tiene influencia debe ser almacenada, (Bo and Huazhong, 2012), (Dussaud et al., 2008). La Figura 3 destaca el *stencil* asociado a algunos de los puntos del modelo de velocidades desde una vista superior. En ellos, se involucran términos de la zona PML, lo cual impediría su reconstrucción. Estos puntos, como se muestra en la Figura 3 son almacenados para hacer la reconstrucción completa del campo.

A pesar de que esta estrategia es una de las más eficientes en términos de manejo de memoria, la cantidad de memoria empleada para almacenar toda la energía que alcanza los bordes también puede ser muy alta. Para el modelo de velocidad mostrado en la tabla 1 y asumiendo que se usan diferencias finitas de 8º orden espacial, las fronteras pueden alcanzar hasta 557.7 GB a 50 Hz.

Además de los potenciales problemas de memoria que puede tener esta estrategia, es necesario analizar el error numérico debido a la reconstrucción. Aunque la manipulación algebraica de la ecuación de onda garantiza que el proceso es completamente invertible, la implementación y el redondeo necesario para la representación numérica de punto flotante introduce pequeños errores

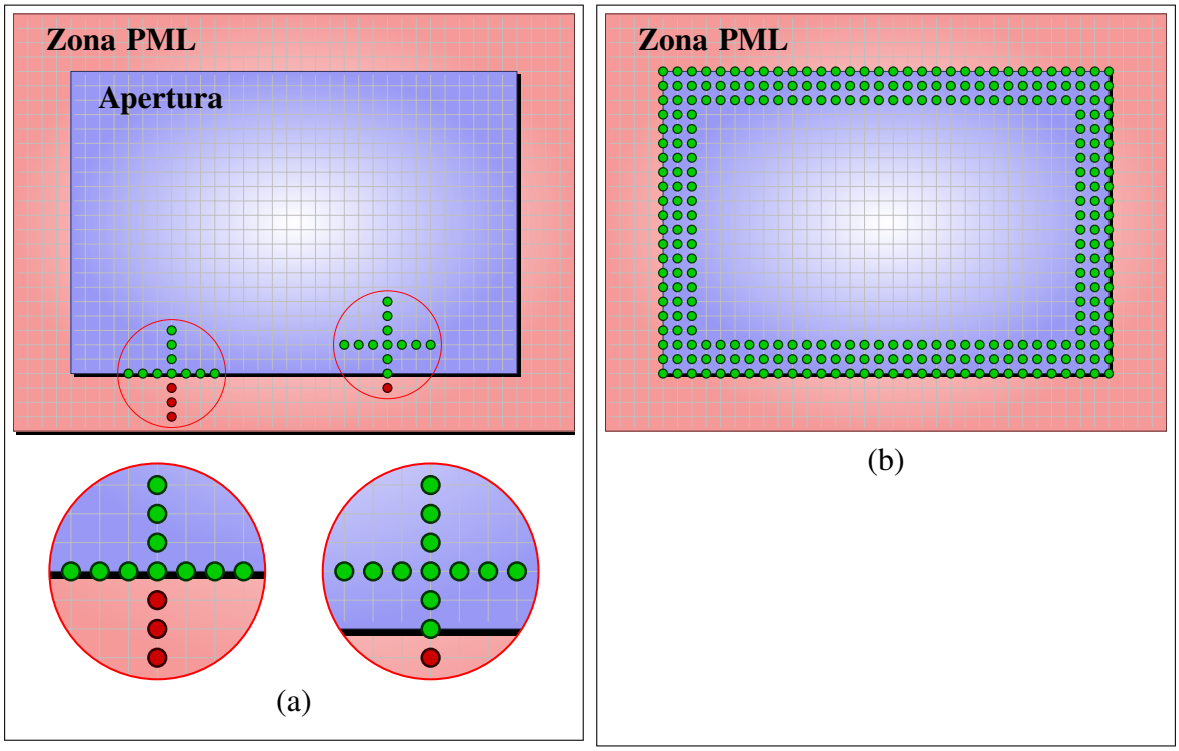


Figura 3. Vista superior del modelo de velocidades. Representación gráfica del *stencil* que relaciona los términos que aportan al cálculo del punto central de algunos puntos del modelo. Se observa que algunos puntos internos requieren de puntos de la zona PML. Zona de frontera que se almacena. El grosor depende del orden de aproximación de las diferencias finitas.

en cada paso de tiempo, que se hacen más evidentes en los primeros pasos de tiempo.

2.3.4. Estrategia 3: Uso de fronteras aleatorias. La técnica de fronteras aleatorias propuesta por Clapp (2009) extiende el modelo de velocidades con valores aleatorios. El objetivo de la técnica es generar un patrón aleatorio de reflexión de la energía que alcanza el borde del modelo. Esto hace que el campo de la fuente reflejado no sea coherente con el campo de los receptores, generando en la imagen final ruido aleatorio; así mismo la energía no es disipada lo cual permite invertir el proceso de modelado con un esquema estable. Esto último reduce significativamente la memoria requerida porque ya no es necesario almacenar las fronteras y el proceso de recómputo

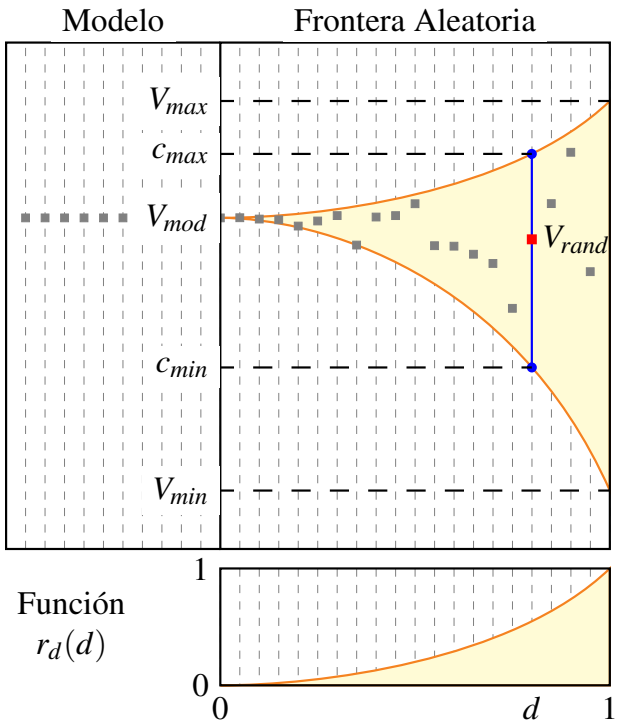


Figura 4. Límites para los valores que puede tomar la velocidad en la frontera aleatoria. Cerca al modelo las variaciones son pequeñas, pero van aumentando de acuerdo con la función $rd(d)$.

del campo se hace únicamente con el valor del campo en los dos últimos pasos de tiempo. Posteriormente Fletcher and Robertsson (2011) propuso una forma generalizada del método donde los valores de velocidad de la frontera se cambian en periodos regulares. Esto permite hacer una reducción adicional en la coherencia del frente de onda reflejado en la frontera aleatoria. A pesar de la gran reducción de memoria requerida, la calidad de la imagen se ve comprometida debido al error numérico, de la misma naturaleza que en la estrategia 2, y el error debido al patrón aleatorio de la energía reflejada.

2.3.4.1. Generación de los números aleatorios. La Figura 4 muestra gráficamente los valores que puede tomar la velocidad en la frontera aleatoria. Todos los valores aleatorios de velocidad generados pertenecen al intervalo $I = \left(V_{min}, V_{max}\right)$, donde V_{max} no debe exceder el valor máximo determinado por la condición de estabilidad y V_{min} no debería llegar a valores que generen problemas de dispersión numérica. Así mismo, cada punto específico estará acotado al subintervalo $\left[c_{min}, c_{max}\right] \subseteq I$ que varía según la distancia al borde del modelo (d). Esto permite que las variaciones de velocidad sean suaves al comienzo de la frontera aleatoria, pero más fuertes cerca al borde externo de la frontera. El ritmo con el que el intervalo crece lo determina la función $r_d(d)$, la cual va de cero a uno en el rango de la frontera aleatoria. De esta forma se define

$$c_{min} = (1 - r_d(d))V_{mod} + r_d(d)V_{min} \tag{31}$$

$$c_{max} = (1 - r_d(d))V_{mod} + r_d(d)V_{max}, \tag{32}$$

donde V_{mod} es el valor original de la velocidad en ese punto. Así mismo si se define R como un valor aleatorio entre cero y uno con distribución uniforme, V_{rand} se puede definir como la suma ponderada entre c_{min} y c_{max} así:

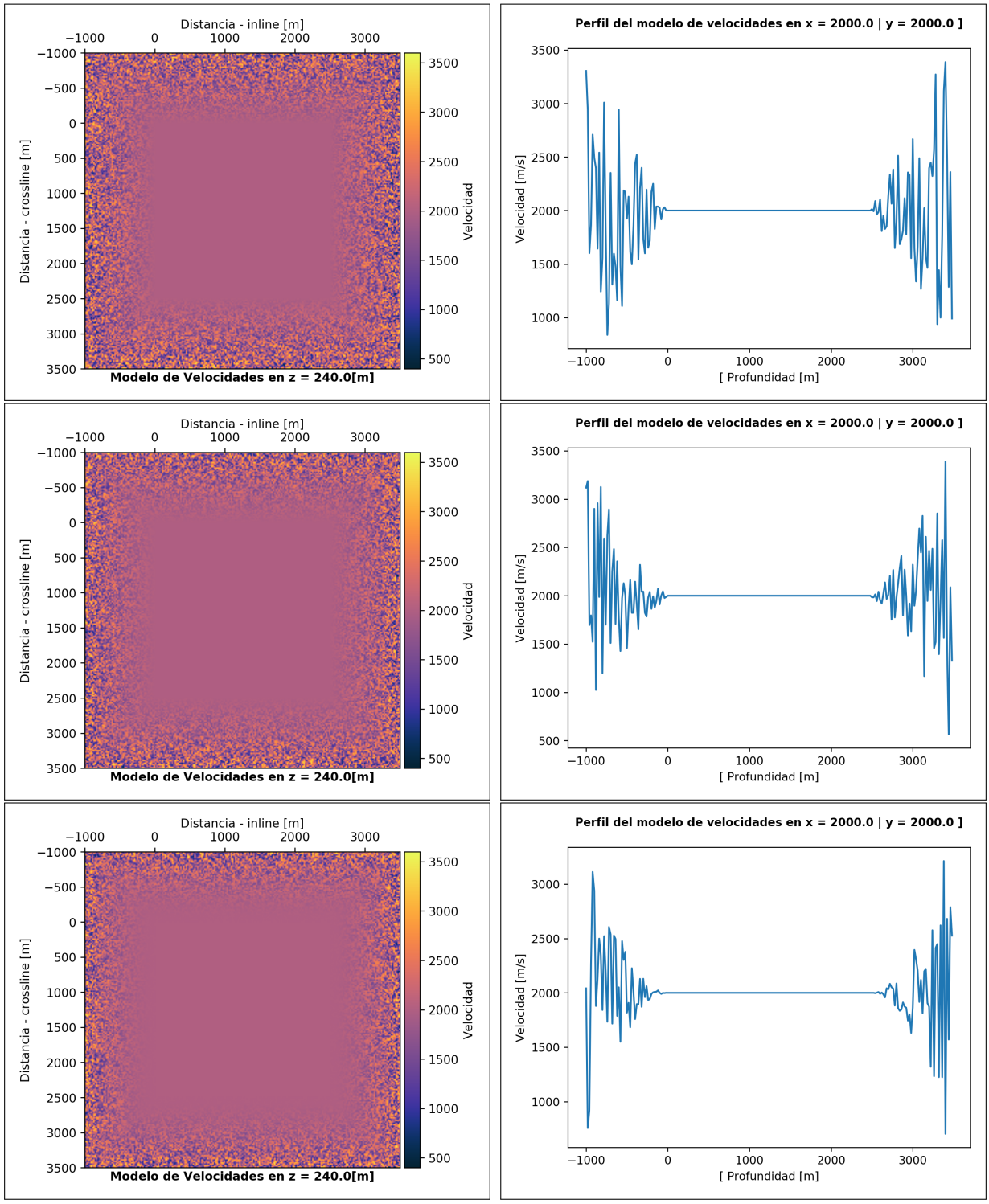


Figura 5. Modelos de velocidades generados con diferentes funciones $r_d(d)$. De arriba hacia abajo: lineal, exponencial y cuadrática. Todos los modelos tienen $V_{mod} = 2000$ [m/s], $V_{min} = 400$ [m/s] y $V_{max} = 3600$ [m/s]. Los paneles del lado izquierdo son cortes del modelo 3D hechos en $z = 240$ [m] y las líneas de los paneles del lado derecho fueron tomadas de $x = 2000$ [m], $y = 2000$ [m].

$$V_{rand} = R\,c_{max} + (1-R)c_{min} \tag{33}$$

$$V_{rand} = \Big[1 - r_d(d)\Big]V_{mod} + r_d(d)\Big[\big(1-R\big)V_{min} + R\,V_{max}\Big] \tag{34}$$

Se puede notar que esta expresión mantiene el valor original de la velocidad cuando $r_d(d) = 0$; así mismo, cuando $rd(d) = 1$, el valor generado tiene el mayor rango posible en el intervalo I.

2.3.4.2. Parámetros de la implementación. Durante la implementación, se trabajaron los siguientes parámetros que modifican la forma como trabaja la frontera aleatoria:

- `rd_type` que permite seleccionar la función con la que crece el intervalo de generación de números aleatorios $r_d(d)$ entre las siguientes opciones:

$$\begin{aligned} &\text{Lineal:} && rd(d) = d \\ &\text{Exponencial:} && rd(d) = \frac{1-e^d}{1-e^1} \\ &\text{Cuadrática:} && rd(d) = d^2 \end{aligned} \tag{35}$$

  La Figura 5 muestra algunos modelos generados para diferentes funciones $r_d(d)$.

- `rand_mode` modifica los valores V_{max} y V_{min} según las siguientes opciones predefinidas:

Tabla 2
Modificación del rango de valores aleatorios mediante le parámetro `rand_mode`.

`rand_mode`	V_{min}	V_{max}
0	0	$V_{MaxEstable}$
1	V_{Nyq}	$V_{MaxEstable}$
2	$4 * V_{Nyq}$	$V_{MaxEstable}$
3	Máximo rango simétrico respecto a V_{mod}	

$$V_{MaxEstable} = \frac{2\,\text{mín}(\Delta x, \Delta y, \Delta z)}{\sqrt{3}\Delta t \sqrt{-C_0 + \sum_{l=1}^{N/2} C_l (-1)^{l+1}}}$$

$$V_{Nyq} = 2\,pplo\,f\ \text{máx}(dx, dy, dz)$$

- `ks_rand` fija el periodo para cambiar los valores aleatorios de la frontera según lo propuesto por Fletcher and Robertsson (2011). Específicamente establece la cantidad de pasos de tiempo entre un cambio y otro.
- `L` que permite variar el ancho de la frontera aleatoria.

2.4. Implementación en GPU

Las estrategias de implementación del algoritmo RTM se desarrollaron usando CUDA-C para una GPU Nvidia. En esta sección se describe los aspectos más importantes de la implementación.

2.4.1. Arquitectura GPU. Las GPU son dispositivos de cómputo paralelo diseñadas para operaciones de video como *image rendering*. Estos dispositivos son sistemas *many-core*

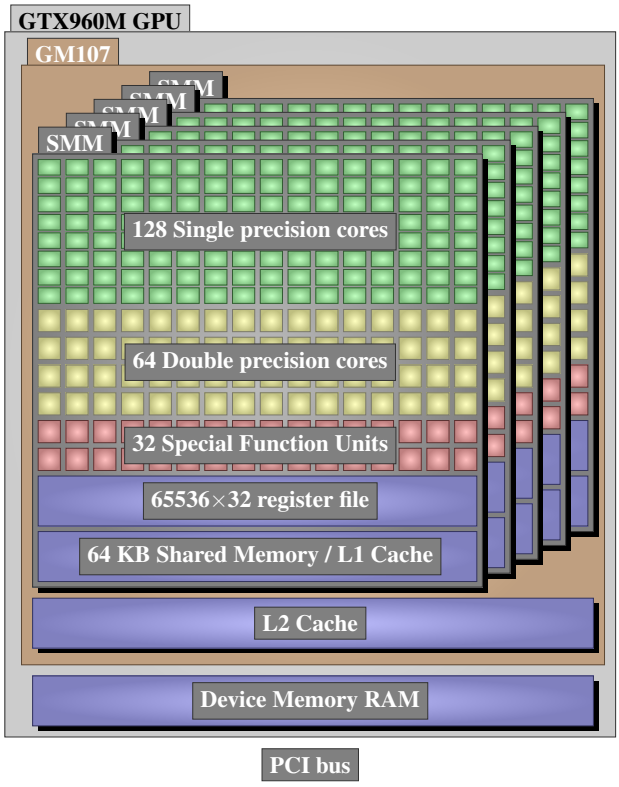


Figura 6. Representación de la arquitectura de la GPU. Se resalta la agrupación de los elementos de cómputo y de almacenamiento

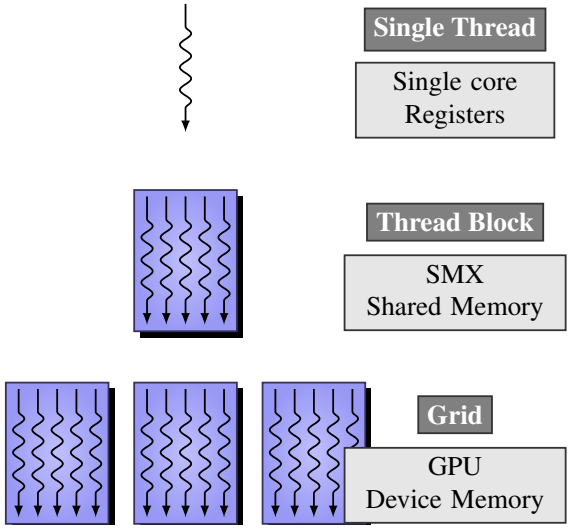


Figura 7. La jerarquía de hilos en la GPU controla el acceso de los mismos a diferentes recursos de hardware.

basados en arquitecturas *Single Instruction Multiple Thread* que han sido adaptados para computación científica. Hoy en día aplicaciones de propósito general pueden ser desarrolladas usando estos dispositivos con modelos de programación y API como CUDA y OpenCL.

Las GPU agrupan miles de elementos de cómputo en un chip lo cual añade nuevos niveles de jerarquía en la memoria como se muestra en la figura 6. Estos elementos representan más capacidad de cómputo que una CPU convencional, pero requieren transferencias de datos óptimas para emplear todo su potencial. La información llega a la GPU desde la CPU a través del puerto PCI y es almacenada en la memoria RAM del dispositivo. Ahí los datos son accedidos por los núcleos de

cómputo al pasar a través de los niveles de caché L1 y L2.

El modelo del programador, establece que los elementos de procesamiento son utilizados al lanzar una gran cantidad de hilos en la GPU. La GPU administra los hilos creando un proceso segmentado entre cómputo y accesos a memoria. Los hilos tienen su propia agrupación jerárquica, lo cual también limita los recursos de la GPU a los cuales pueden acceder. En la figura (7) se observa que cada hilo tiene acceso privado a su propio grupo de registros de propósito general; así mismo grupos de hilos, llamados bloques, pueden acceder a los recursos de un único *Streaming Multiprocessor (SMX)*, incluyendo la memoria compartida; finalmente el *grid* compuesto por todos los bloques puede acceder a todos los recursos de la GPU, incluyendo su memoria RAM.

El modelo de abstracción de hardware dado por las API permite a los programadores entender los componentes de la GPU y desarrollar aplicaciones eficientes. Así mismo los fabricantes de GPU dan un conjunto de recomendaciones para mejorar el desempeño de las aplicaciones, especialmente en lo que respecta a los accesos a memoria. Estas recomendaciones incluyen la reducción de transferencias entre CPU y GPU pero debido a la gran cantidad de memoria requerida por el algoritmo RTM se puede hacer uso de la memoria de la CPU o ajustar las estrategias para usar sólo la memoria disponible en la GPU.

2.4.2. Programación. Las tres estrategias presentadas fueron implementadas sobre una GPU Nvidia GTX960M. La ecuación acústica con densidad constante fue solucionada mediante diferencias finitas y se implementó CPML como método de frontera absorbente para evitar las reflexiones artificiales en el borde del modelo.

La base de cualquier estrategia es la función para avanzar un paso de tiempo en el proceso de modelado. Cada paso de tiempo emplea un esquema explícito para extrapolar en tiempo los campos de onda mediante la Ecuación 19. En este tipo de esquemas, cada punto espacial es calculado de forma independiente, lo cual favorece su implementación en plataformas paralelas. La implementación desarrollada, divide el paso de tiempo en dos *kernels*: una para solucionar la ecuación de onda y otra para los puntos que pertenecen a la frontera absorbente.

Antes de iniciar el procesamiento en la GPU, los datos deben ser transferidos a ésta. La implementación paralela hace transferencias de datos sólo al principio y al final de la migración de un disparo para minimizar la transferencias entre CPU y GPU. La cantidad de memoria disponible para procesar cada disparo en la GPU GTX960M, es 2 GB.

2.5. Resultados

En esta sección se analizan los resultados de la implementación de cada una de las estrategias y se estima el error relativo al variar algunos parámetros de ejecución. Así mismo se miden los recursos computacionales empleados y el tiempo de ejecución.

2.5.1. Estrategia 1. La estrategia 1 es, por definición, la que tiene menor cantidad de factores de error. Por ello fue tomada como referencia para comparar resultados de las siguientes estrategias. Se usó el modelo de velocidades de la SEG-EAGE, cuyos parámetros se pueden ver en la Tabla 3 y algunos cortes del campo de velocidades en la Figura 8. Sobre este modelo se hizo una adquisición con los parámetros de la Tabla 4. La Figura 9 muestra algunos cortes del resultado

de migrar el modelo SEG-EAGE con los parámetros de la Tabla 5. En el resultado obtenido se observan bien definidas las fronteras del cuerpo de sal, incluso el borde inferior del mismo. Para migrar este dato, se requirió ajustar el parámetro `ks_store` que determina la separación de los puntos de control. La memoria disponible, permitió almacenar 53 puntos de control distribuidos uniformemente a `ks_store` $= 48$ pasos de tiempo.

Tabla 3
Parámetros del modelo de velocidades SEG-EAGE

Parámetros del modelo	
N_x, N_y, N_z [puntos]	$(169, 169, 50)$
$\Delta x, \Delta y, \Delta z$ [m]	$(80, 80, 80)$
Tamaño [m]	$(13440, 13440, 3920)$
V_{min} [m/s]	1499.92
V_{max} [m/s]	4482

Tabla 4
Parámetros de la adquisición sobre el modelo SEG-EAGE

Parámetros de la adquisición	
SI, SLI [m]	$800, 800$
RI, RLI [m]	$80, 80$
Tamaño del *patch* [m]	4000×4000
Número de disparos	144
Tiempo de grabación [s]	5
Periodo de muestreo [ms]	4

Tabla 5
Parámetros de la migración del dato completo sobre el modelo SEG-EAGE. Los parámetros fueron ajustados para permitir la comparación de todas las estrategias empleando la memoria disponible en la GPU GTX960M.

Extensión del modelo	
left [m]	640
right [m]	640
back [m]	640
front [m]	640
top [m]	3000
bottom [m]	640

Parámetros de ejecución	
`pplo`	5
`Lpml` [puntos]	16
`dt`(usuario) [s]	0.002
`fq` [Hz]	7
`order`	6
`abc`	1, 1, 1, 1, 1, 1

Apertura [m]	
left	160
right	160
back	160
front	160
top	160
bottom	10000

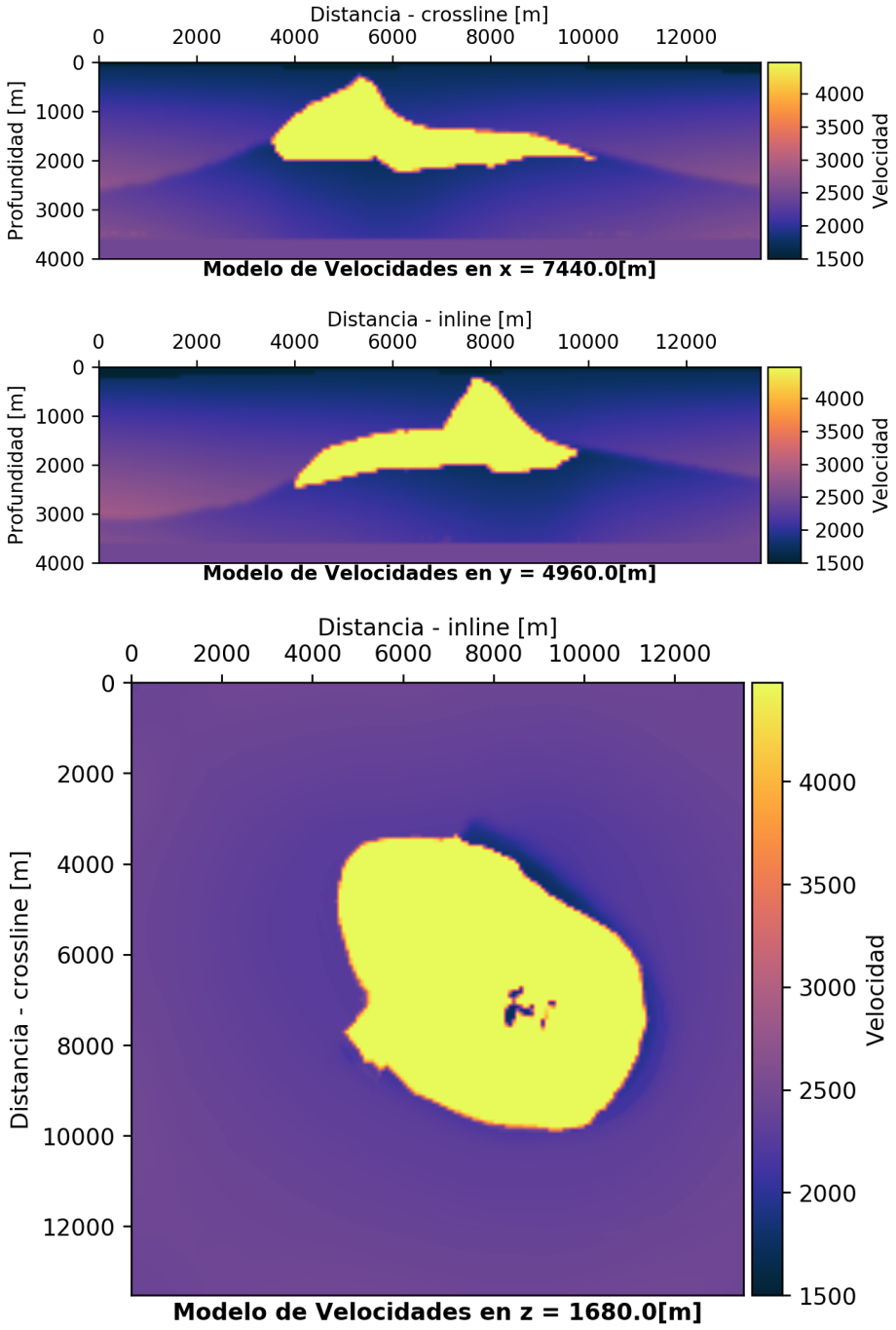


Figura 8. Modelo de velocidades SEG-EAGE. El panel superior muestra un corte en el punto $x = 7440$ [m]. El segundo panel hace un corte en el punto $y = 4960$ [m] y finalmente el tercer panel muestra un corte en $z = 1680$ [m].

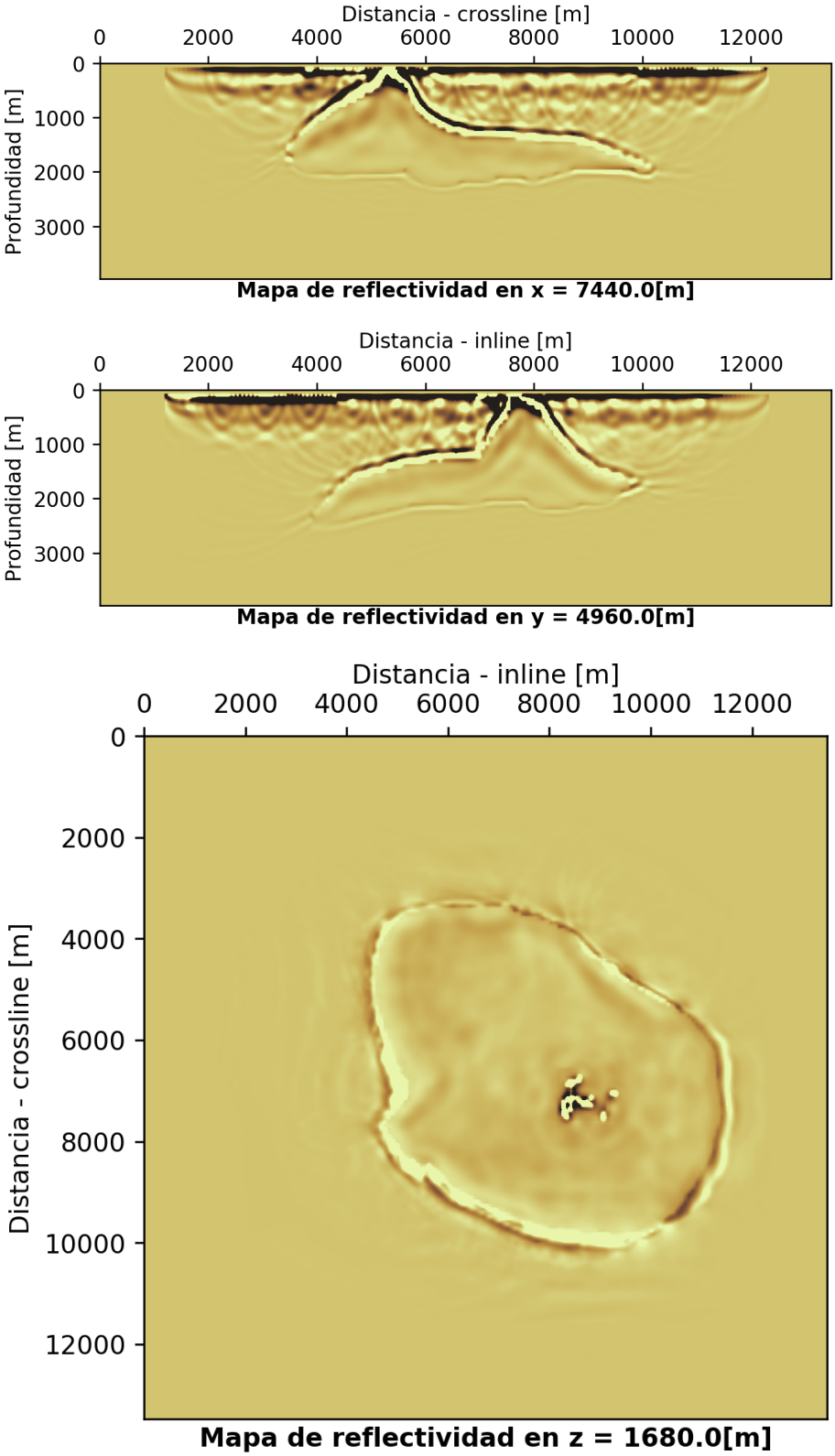


Figura 9. Mapa de reflectividad (Imagen migrada). Realizado mediante la estrategia 1. El panel superior muestra un corte en el punto $x = 7440$ [m]. El segundo panel hace un corte en el punto $y = 4960$ [m] y finalmente el tercer panel muestra un corte en $z = 1680$ [m].

2.5.2. Estrategia 2. Esta estrategia no tiene un parámetro para ajustar la memoria requerida. Se migró el modelo SEG-EAGE con los mismos parámetros de migración de la estrategia 1 (Tabla 5). Algunos cortes del resultado se aprecian en la Figura 10. Adicionalmente, la Figura 11 muestra la diferencia entre los resultados de la estrategia 1 y la estrategia 2. La causa de la diferencia entre las dos soluciones se debe al error numérico de la representación de punto flotante que impide que la retropropagación del campo de la fuente sea exacta. Este error se estimó mediante la norma L2 normalizada sobre todo el cubo en: $2.681954e-06$. Se observa en la Figura 11 que el error se concentra en la región somera del modelo debido a que, como se verá en la siguiente subsección, el error en la reconstrucción del campo de la fuente es muy alto en los primeros tiempos del modelado.

2.5.2.1. Error numérico de la reconstrucción estrategia 2. Para evaluar el error numérico en esta estrategia, se empleó un modelo de velocidad constante con los parámetros de la Tabla 6. Se restringió el algoritmo para hacer únicamente modelado y reconstrucción del campo de la fuente y finalmente se utilizó la norma L2 normalizada para comparar el campo de la fuente propagado (empleado como referencia) y el campo de la fuente reconstruido. En ese orden de ideas, se analizó cómo afectan al error los siguientes factores:

- Posición de la fuente: Distancia al borde del modelo.
- Representación numérica de punto flotante: *float* o *double*.
- Orden de aproximación de las diferencias finitas.

La Figura 12 muestra el error cuando el modelado se realiza con datos de punto flotante

Tabla 6
Parámetros del modelo de velocidad constante.

Parámetros del modelo	
N_x, N_y, N_z [puntos]	$(100, 100, 100)$
$\Delta x, \Delta y, \Delta z$ [m]	$(10, 10, 10)$
Tamaño [m]	$(990, 990, 990)$
V_{min} [m/s]	1500
V_{max} [m/s]	1500

Tabla 7
Parámetros del modelado sobre el modelo de velocidad constante.

Parámetros del modelado	
`pplo`	10
`Lpml` [puntos]	2
`dt`(usuario) [s]	0.0009
`fq` [Hz]	15
`abc`	1, 1, 1, 1, 1, 1
`itmax` [pasos]	1333

de precisión sencilla y doble con la fuente ubicada en el centro del modelo. La tendencia de las dos gráficas es muy similar. Es evidente que la representación de precisión doble tiene un error considerablemente menor a la precisión sencilla debido principalmente al tamaño de la mantisa (23 bits en precisión sencilla y 52 bits en precisión doble). Esta diferencia, se mantiene aún si la fuente está en el borde del modelo, como se muestra en la Figura 13.

En cuanto a las tendencias de las Figuras 12 y 13, se observa que el error es menor en los últimos pasos de tiempo, debido a que éste se acumula con cada paso que se reconstruye hacia atrás. El valor de la métrica se mantiene constante durante la mayor parte de la reconstrucción y tiene un valor muy alto en los tiempos cercanos a cero debido a que los errores acumulados no permiten que la energía colapse en el punto donde se localiza la fuente. La Figura 16 presenta una comparación de *snapshots* del campo de la fuente propagada y reconstruida en diferentes tiempos para el caso donde la fuente se localiza en el centro. De igual forma la Figura 17 muestra *snapshots* para cuando la fuente se ubica en la superficie del modelo.

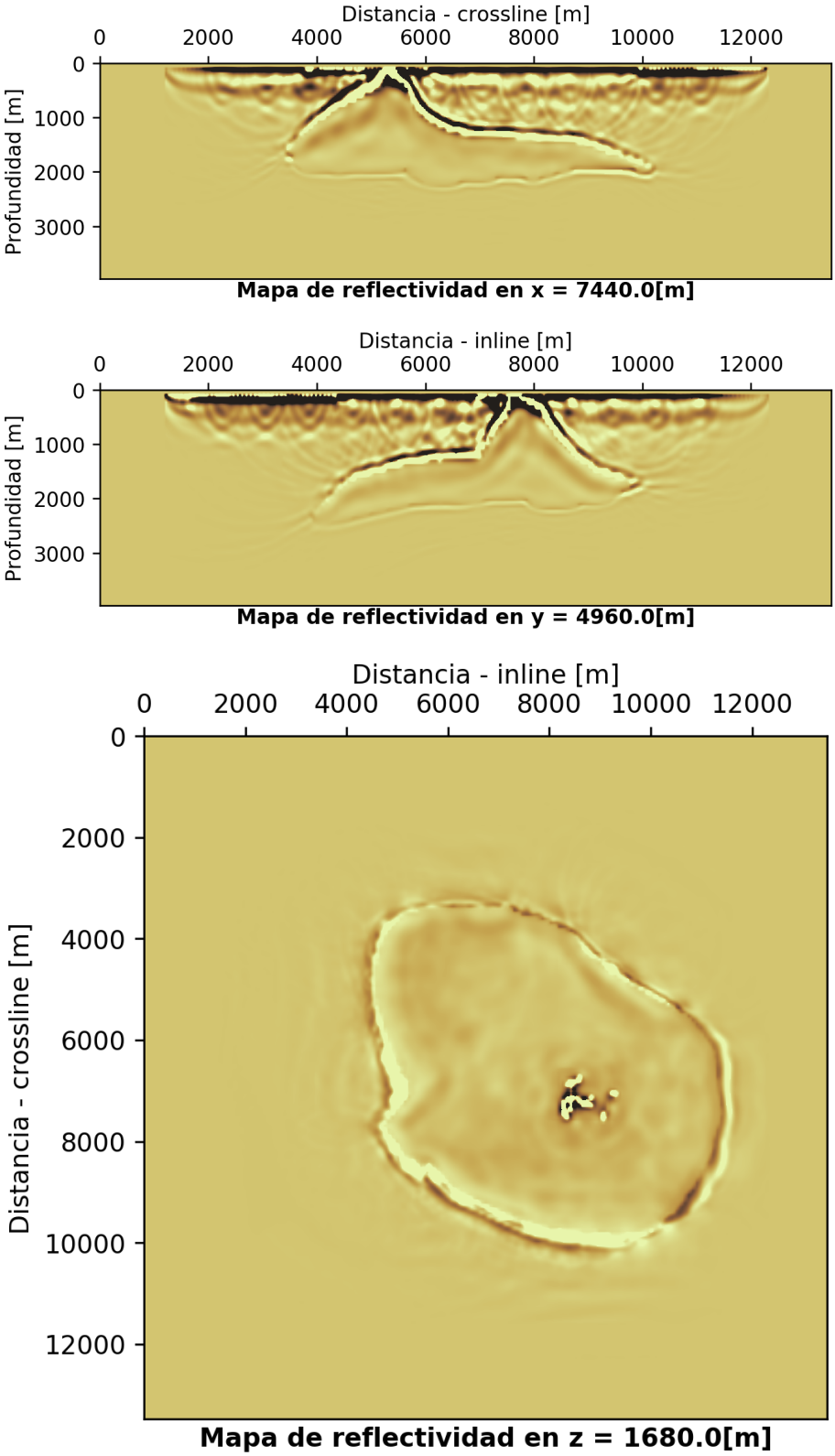
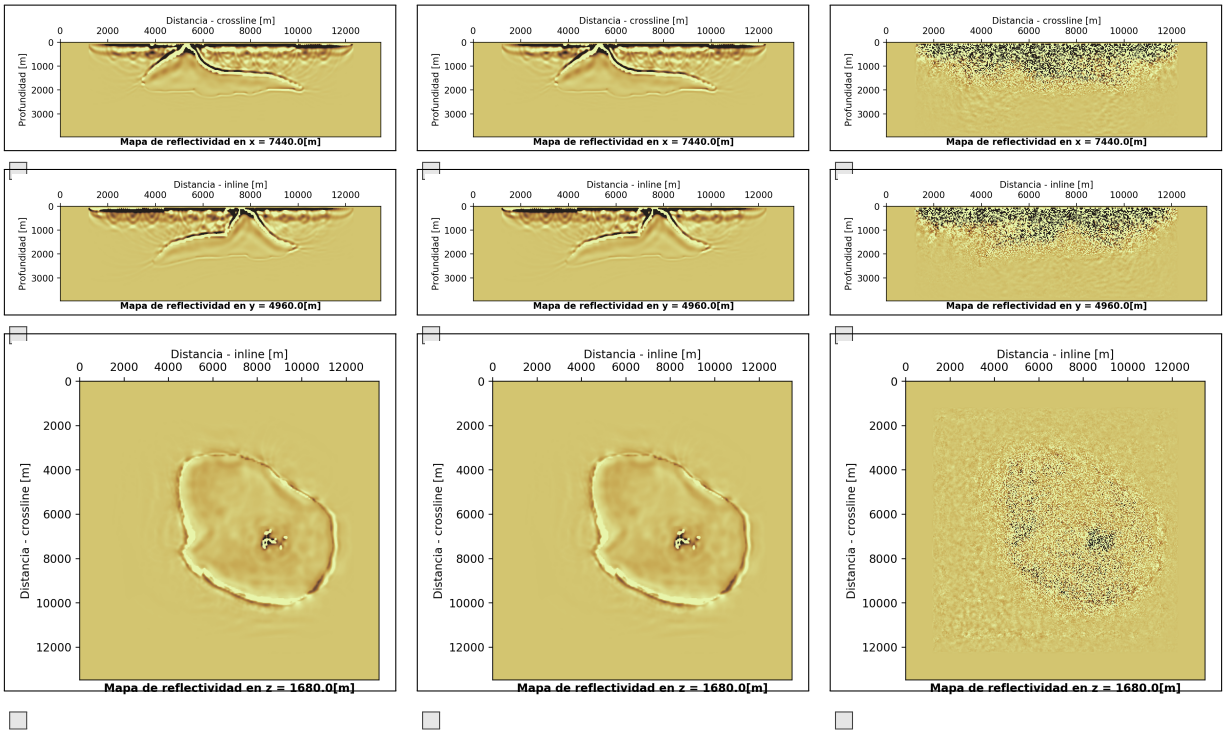


Figura 10. Mapa de reflectividad (Imagen migrada). Realizado mediante la estrategia 2. El panel superior muestra un corte en el punto $x = 7440$ [m]. El segundo panel hace un corte en el punto $y = 4960$ [m] y finalmente el tercer panel muestra un corte en $z = 1680$ [m].



(a) Resultado Estrategia 1 | **(b) Resultado Estrategia 2** | **(c) Diferencia** $\times 1000000$

Figura 11. Comparación de resultados al migrar un dato completo sobre el modelo SEG-EAGE con las estrategias 1 y 2. Los paneles izquierdos corresponden al resultado de la estrategia 1 que se usó como referencia. Los paneles de la columna central son resultado de utilizar la estrategia 2 y los paneles de la derecha son la diferencia entre los dos paneles iniciales con un factor de 1000000. En todas las columnas, los paneles superiores muestran el corte en el punto $x = 7440$ [m]. Los paneles intermedios hacen un corte en el punto $y = 4960$ [m] y finalmente los paneles inferiores muestran un corte en $z = 1680$ [m].

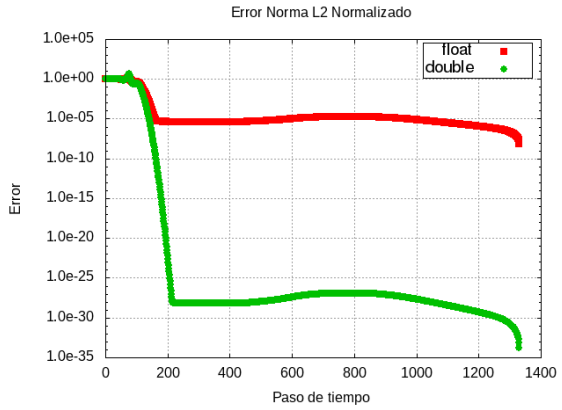


Figura 12. Error del campo de la fuente reconstruido hacia atrás versus el campo propagado. La ubicación de la fuente fue el punto central de un modelo de velocidad constante.

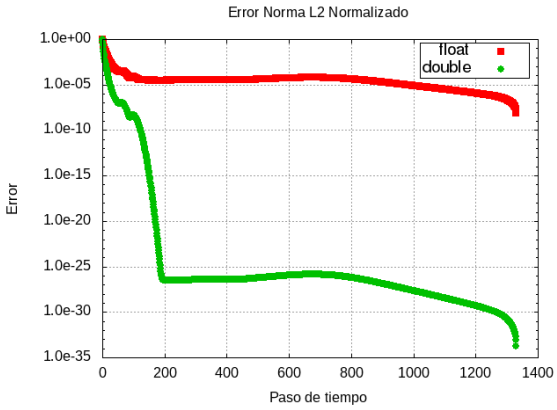


Figura 13. Error del campo de la fuente reconstruido hacia atrás versus el campo propagado. La ubicación de la fuente fue el centro de la cara superior de un modelo de velocidad constante.

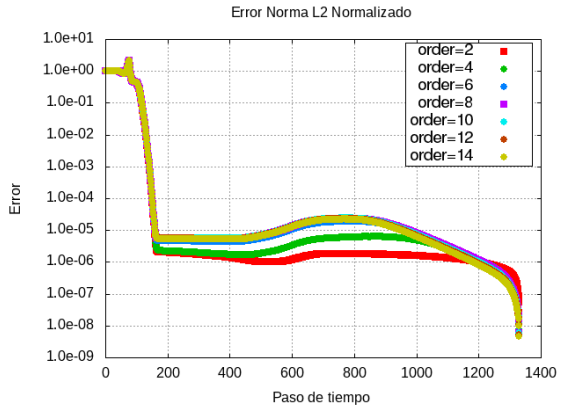


Figura 14. Efecto del orden de aproximación de las diferencias finitas. Error del campo de la fuente reconstruido hacia atrás versus el campo propagado. La ubicación de la fuente fue el punto central de un modelo de velocidad constante.

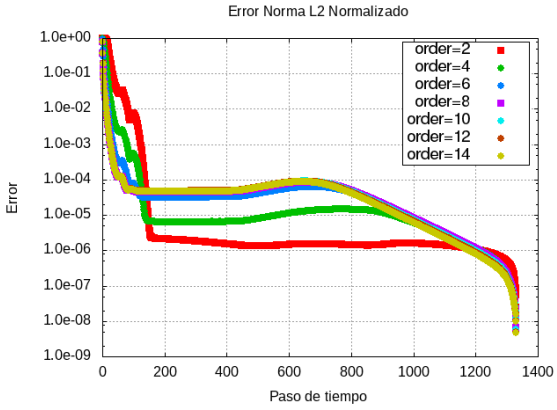


Figura 15. Efecto del orden de aproximación de las diferencias finitas. Error del campo de la fuente reconstruido hacia atrás versus el campo propagado. La ubicación de la fuente fue el centro de la cara superior de un modelo de velocidad constante.

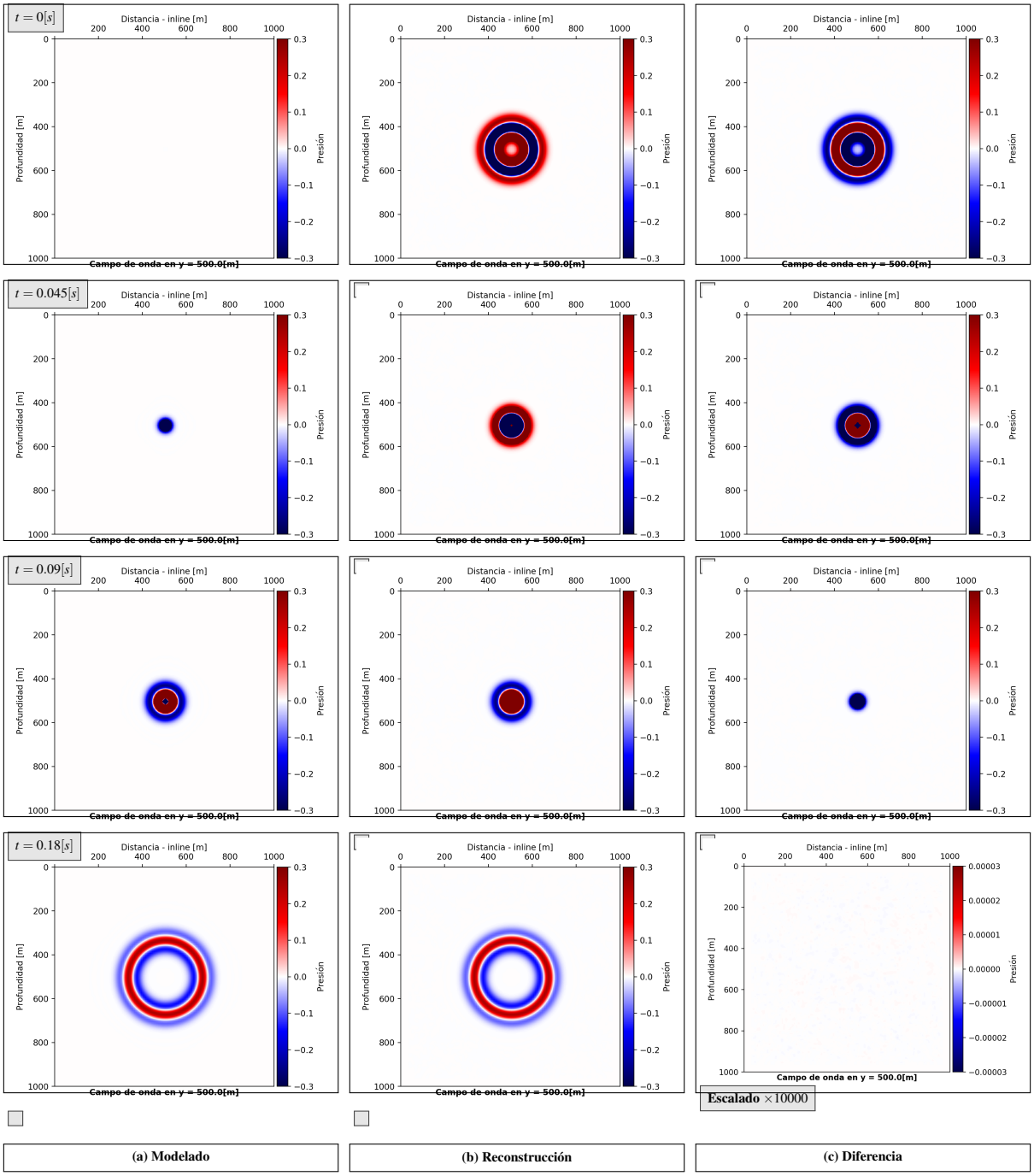


Figura 16. Comparación del campo de onda de la fuente contrastando la versión propagada contra la versión reconstruida desde el tiempo final con los dos últimos *snapshots* y las fronteras. En los paneles de la izquierda, se observa el campo modelado, en los paneles centrales se observa el campo reconstruido y en los paneles de la derecha aparece la diferencia entre estos dos campos. En todos los casos se compara un corte hecho para $x = 500$ [m] y la fuente se ubicó en el centro del modelo. Los parámetros del modelo y la propagación están en las Tablas 6 y 7.

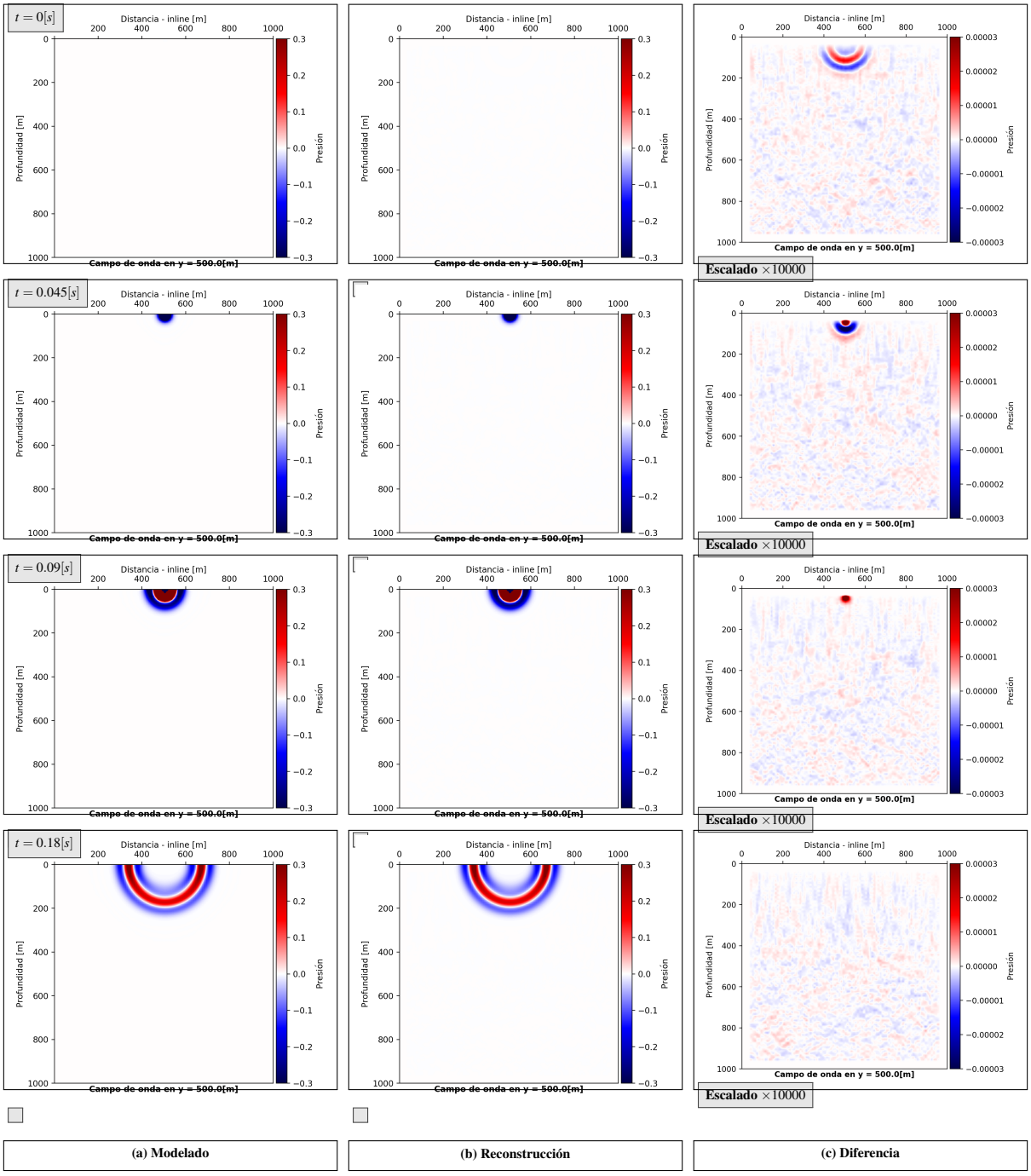


Figura 17. Comparación del campo de onda de la fuente contrastando la versión propagada contra la versión reconstruida desde el tiempo final con los dos últimos *snapshots* y las fronteras. En los paneles de la izquierda, se observa el campo modelado, en los paneles centrales se observa el campo reconstruido y en los paneles de la derecha aparece la diferencia entre estos dos campos. En todos los casos se compara un corte hecho para $x = 500$ [m] y la fuente se ubicó en la cara superior del modelo. Los parámetros del modelo y la propagación están en las Tablas 6 y 7.

El último parámetro que se comparó fue el orden de aproximación de las diferencias finitas. Las Figuras 14 y 15 realizan el mismo experimento con fuentes en el centro del modelo y en un borde para diferencias finitas que van desde orden 2 hasta orden 14. Se pueden analizar diferentes momentos desde el tiempo final hasta el inicio. En los últimos pasos de tiempo, la menor aproximación da un error mayor y rápidamente cambia esa tendencia que se mantiene hasta el incremento en los tiempos iniciales. El menor orden de aproximación tiene, en la mayor parte del tiempo de modelado, un error menor debido a que su cálculo implica un menor número de operaciones y por ende menor cantidad de aproximaciones por redondeo. En los tiempos iniciales, el comportamiento es diferente según la posición de la fuente. Para la fuente en el centro del modelo, el error es grande en magnitud, y sin importar el orden de aproximación, el comportamiento es el mismo. En cambio para la fuente en el borde superior del modelo, el error se mantiene en niveles mucho menores y nuevamente la aproximación de menor orden tiene el mayor error. La fuente en el borde del modelo, hace que la frontera almacenada contenga más energía(información) de la fuente, por ello el error es menor comparado con el experimento con la fuente en el centro.

2.5.3. Estrategia 3. La implementación permitió migrar el modelo SEG-EAGE con los mismos parámetros de las dos estrategias anteriores. El resultado se observa en la Figura 19, donde se presentan los mismos cortes que en las estrategias anteriores. Adicionalmente la Figura 18 muestra la diferencia respecto a la estrategia 1, donde el error con la norma L2 normalizada se estimó en : $3.970529e-03$. Se observa que el error es mayor en la superficie, debido a que en esta zona se encuentran las fuentes y el patrón de reflexión en la frontera aleatoria tiene mayor energía allí.

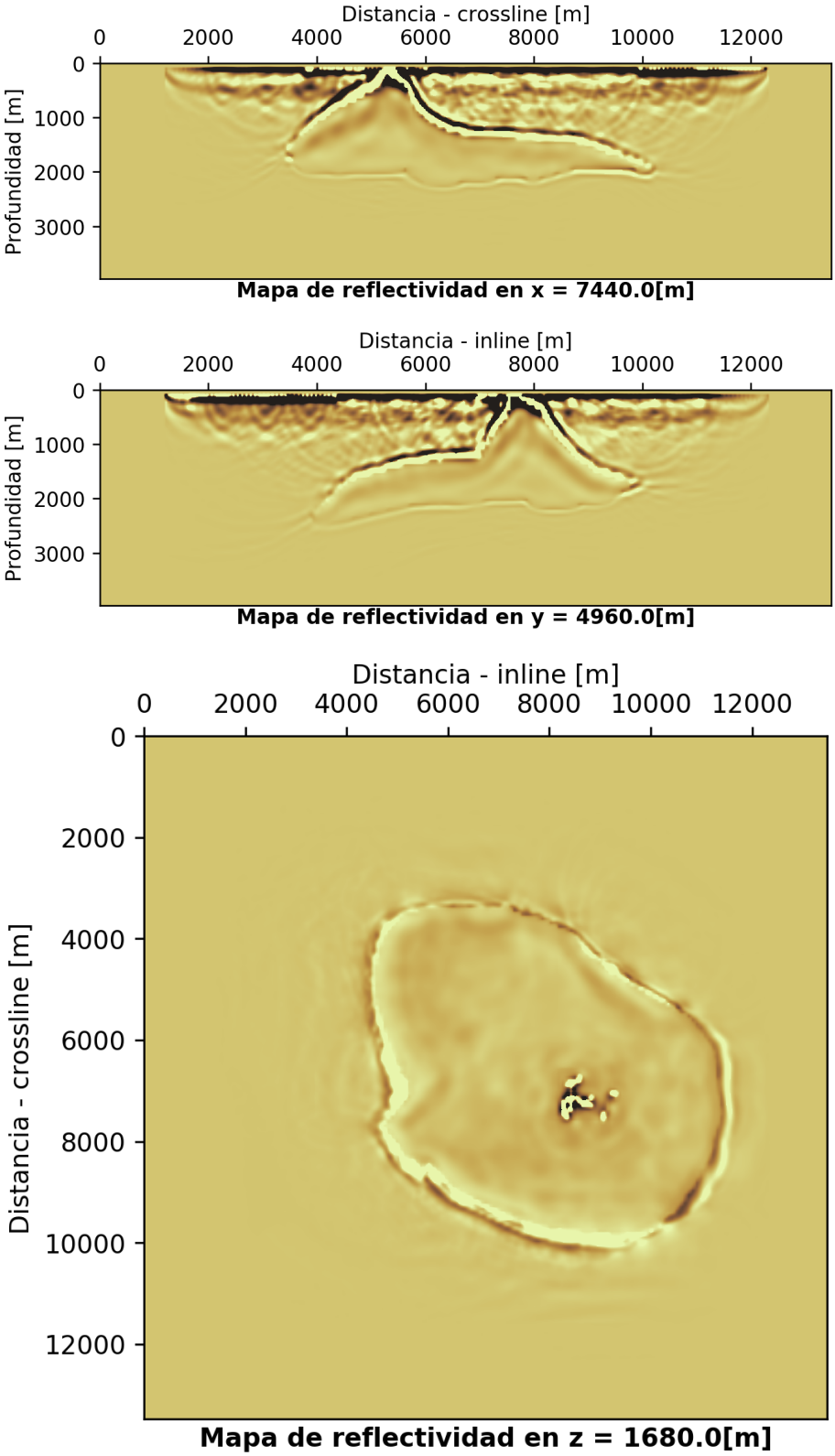
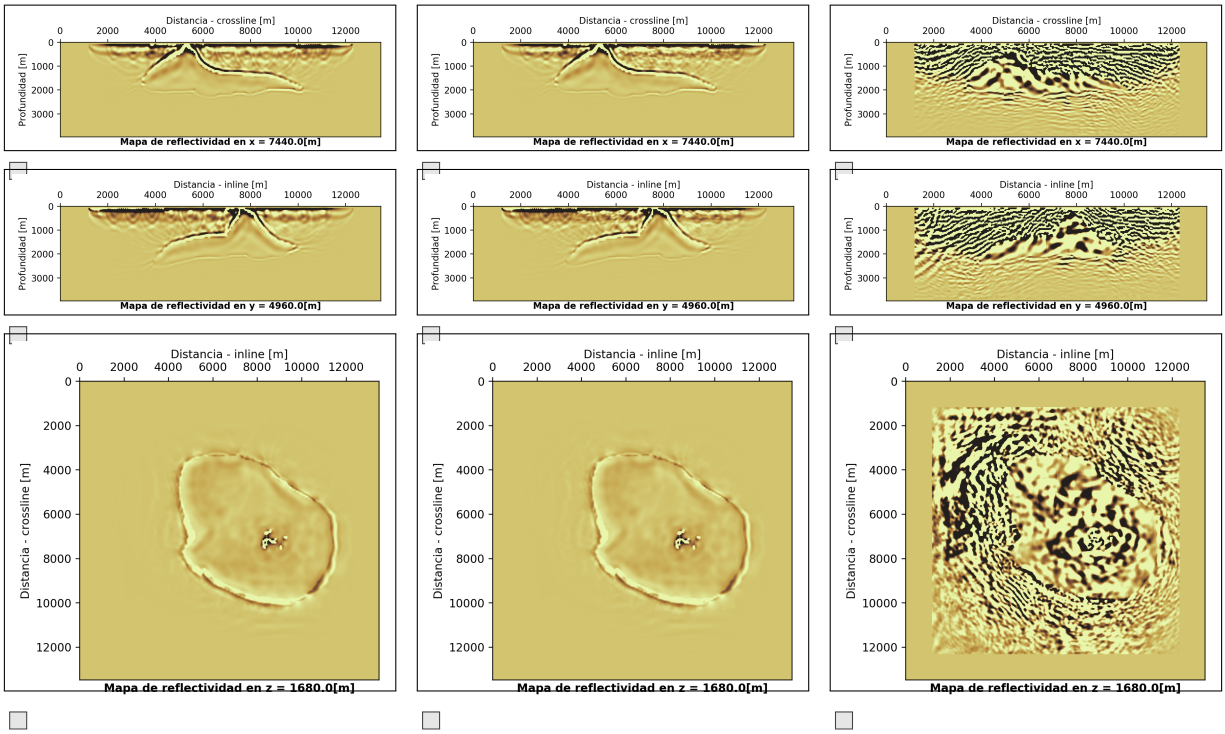


Figura 19. Mapa de reflectividad (Imagen migrada). Realizado mediante la estrategia 3. El panel superior muestra un corte en el punto $x = 7440$ [m]. El segundo panel hace un corte en el punto $y = 4960$ [m] y finalmente el tercer panel muestra un corte en $z = 1680$ [m]. Los parámetros empleados para la ejecución de la estrategia 3 fueron: `rand_mode` $= 3$, `L` $= 16$, `ks_rand` $= 5000$ y `rd_type` $= 2$.



(a) Resultado Estrategia 1 | **(b) Resultado Estrategia 3** | **(c) Diferencia** $\times 1000$

Figura 18. Comparación de resultados al migrar un dato completo sobre el modelo SEG-EAGE con las estrategias 1 y 3. Los paneles izquierdos corresponden al resultado de la estrategia 1 que se usó como referencia. Los paneles de la columna central son resultado de utilizar la estrategia 3 y los paneles de la derecha son la diferencia entre los dos paneles iniciales con un factor de 10000. En todas las columnas, los paneles superiores muestran el corte en el punto $x = 7440$ [m]. Los paneles intermedios hacen un corte en el punto $y = 4960$ [m] y finalmente los paneles inferiores muestran un corte en $z = 1680$ [m]. Los parámetros empleados para la ejecución de la estrategia 3 fueron: `rand_mode` $= 3$, `L` $= 16$, `ks_rand` $= 5000$ y `rd_type` $= 2$.

La implementación de esta estrategia, permitió explorar el efecto de varios parámetros adicionales a los reportados en la literatura. En las siguientes pruebas se presentan resultados de variar los siguientes parámetros:

`rand_mode`**:** Establece los límites de las velocidades aleatorias V_{min} y V_{max} para cada punto.

`ks_rand`**:** Establece el periodo para generar nuevamente los valores aleatorios de la frontera.

`L`**:** Establece el ancho de la frontera aleatoria.

`rd_type`**:** Selecciona la función $r_d(d)$ en sus variantes: lineal, exponencial y cuadrática que controla el incremento del rango de velocidades según la distancia al borde del modelo.

Los parámetros `rand_mode`, `ks_rand`, `L` y `rd_type` son establecidos por el usuario y su implementación se detalla en el Anexo 2.

2.5.3.1. Variación de `rand_mode` y `ks_rand`. En esta sección se explorará experimentalmente la variación de los parámetros `rand_mode` y `ks_rand`. La función $r_d(d)$ será lineal, como en las implementaciones originales de Clapp et al. (2010) y el ancho de la frontera aleatoria será `L` $= 50$ para todas las pruebas. Para estas pruebas se empleó un modelo de velocidad constante con los parámetros de la Tabla 8 y se localizó una fuente de frecuencia central $f_q = 10Hz$ en el centro del modelo.

Tabla 8
Parámetros del modelo de velocidad constante.

Parámetros del modelo	
N_x, N_y, N_z [puntos]	$(250, 250, 250)$
$\Delta x, \Delta y, \Delta z$ [m]	$(20, 20, 20)$
Tamaño [m]	$(4980, 4980, 4980)$
V_{min} [m/s]	2000
V_{max} [m/s]	2000

Tabla 9
Parámetros del modelado sobre el modelo de velocidad constante.

Parámetros del modelado	
`ord`	8
`pplo`	10
`Lpml` [puntos]	50
`dt`(usuario) [s]	0.001
`fq` [Hz]	10
`abc`	1, 1, 1, 1, 1, 1
`itmax` [pasos]	5000

Según los parámetros establecidos, y aplicando la definición de la Tabla 2, el argumento `rand_mode` establecerá el rango de velocidades aleatorias así:

Tabla 10
Modificación del rango de valores aleatorios mediante el parámetro `rand_mode`.

`rand_mode`	V_{min} [m/s]	V_{max} [m/s]
0	0	9057
1	400	9057
2	1600	9057
3	400	3600

$V_{MaxEstable} = 9057$ [m/s]

$V_{Nyq} = 400$ [m/s]

El primer objetivo de la prueba es garantizar que el esquema es estable debido a que se encontró en las pruebas preliminares que la energía del campo aumentaba conforme se avanzaba en el proceso de modelado, aún cuando el algoritmo se mantiene alejado del límite de la condición de estabilidad. Para estudiar la estabilidad se estimó la energía del campo de la fuente mediante la métrica

$$E = \sum_{\forall x,y,z} p_s * p_s. \tag{36}$$

La Figura 20 describe la variación de la métrica E durante el modelado (en la línea roja) y luego durante la reconstrucción (en la línea verde). Se estudian dos casos: uno estable con $L = 50$,

r_d lineal, `rand_mode` $= 0$ y `ks_rand` $= 10000$ y otro inestable con $L = 50$, r_d lineal, `rand_mode` $= 0$ y `ks_rand` $= 150$. En el caso estable, la métrica E sigue la misma tendencia cuando se modela el campo y durante la reconstrucción; además se mantiene en un rango de valores acotados. En el caso inestable, durante el modelado ya se presenta un crecimiento de E en los últimos tiempos y aunque la reconstrucción en un principio trata de volver a los valores acotados, termina cayendo en inestabilidad.

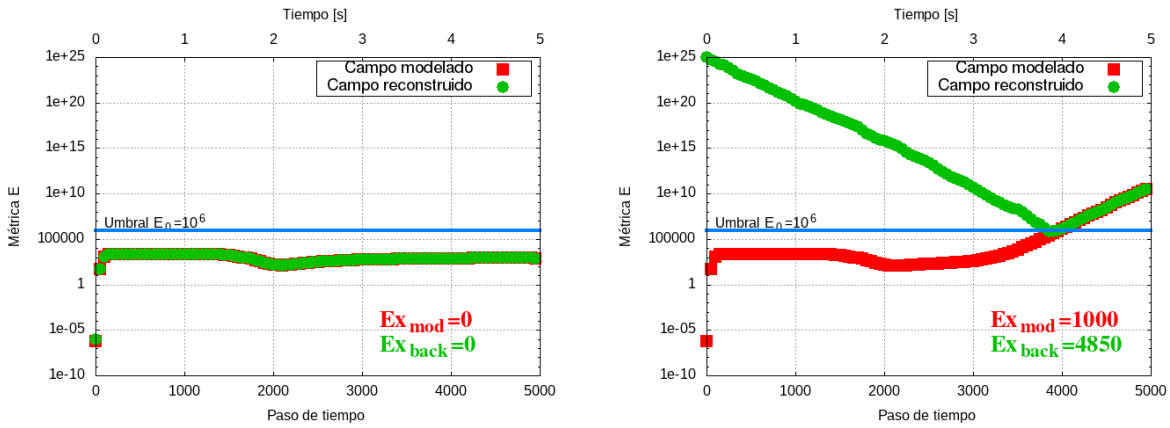


Figura 20. Contraste del comportamiento de la métrica E entre un esquema estable y uno inestable.

En la Figura 20 también se resalta un umbral con el valor de $E_0 = 10^6$, escogido arbitrariamente que permitirá estimar la rapidez con la que el sistema se torna inestable. Los valores Ex_{mod} y Ex_{back} representarán la cantidad de pasos de tiempo cuya métrica E está por encima del umbral durante el modelado y la reconstrucción del campo respectivamente. La Tabla 11 muestra el valor Ex_{mod} y Ex_{back} para los experimentos realizados variando los dos parámetros `rand_mode` y `ks_rand`. Aquí se observa que la acción conjunta de los dos parámetros, compromete la estabilidad del sistema. El peor caso se da cuando el modelo de velocidades se modifica en cada paso de tiem-

po (`ks_rand` = 1) y se tiene el mayor rango de variación para la velocidad con `rand_mode` = 0. Se observa en la columna `rand_mode` = 3 de la Tabla 11, que la inestabilidad continúa incluso cuando se tiene el intervalo más pequeño en el proceso de generación de velocidad aleatoria. Este rango se empleó con el objetivo de mantener una media constante en todos los puntos de la región de la frontera.

Tabla 11
Estabilidad del esquema de diferencias finitas para diferentes valores de `ks_rand` *y* `rand_mode`*. El valor de cada casilla corresponde a la cantidad de pasos de tiempo cuya métrica E que superan el umbral* E_0 *durante el modelado y la reconstrucción del campo de la fuente.*

`ks_rand`	`rand_mode`							
	0		1		2		3	
	Ex_{mod}	Ex_{back}	Ex_{mod}	Ex_{back}	Ex_{mod}	Ex_{back}	Ex_{mod}	Ex_{back}
1	3100	5000	3100	5000	3050	5000	2200	5000
2	3100	5000	3100	5000	3050	5000	2450	5000
4	2950	5000	2950	5000	2850	5000	2550	5000
6	2900	5000	2850	5000	2700	5000	2350	5000
10	2700	5000	2650	5000	2250	5000	2150	5000
20	2450	5000	2350	5000	1700	5000	1600	5000
40	2100	5000	1700	5000	500	1450	400	1850
60	1900	5000	1300	5000	0	0	0	0
100	1450	5000	450	1550	0	0	0	0
150	1000	4850	0	0	0	0	0	0
200	600	3700	0	0	0	0	0	0
250	200	2350	0	0	0	0	0	0
300	0	2150	0	0	0	0	0	0
400	0	950	0	0	0	0	0	0
500	0	0	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0
10000	0	0	0	0	0	0	0	0

El parámetro `rand_mode` modifica el intervalo de valores sobre el cual se selecciona la velocidad aleatoria. Las Figuras 21 y 22 presentan *snapshots* del campo de onda modelado a los 2 y 3 segundos respectivamente, para cada una de las alternativas implementadas con el parámetro `rand_mode`. La región sombreada representa la zona donde se ha extendido el modelo de velocidades para generar la frontera aleatoria.

A los 2 segundos, el frente de onda ha ingresado completamente en la frontera aleatoria; en el interior del modelo solo se observan las reflexiones aleatorias que retornan por el cambio de velocidad de la frontera. Las variaciones suaves dentro de la frontera generan un frente de onda débil en la primera reflexión para cuando `rand_mode` $= 0, 1, 2$. Este frente de onda no aparece cuando `rand_mode` $= 3$ debido a la simetría de V_{max} y V_{min} respecto a la velocidad del borde del modelo, lo cual mantiene una media constante en los valores aleatorios de velocidad generados al interior de la frontera. La deformación del frente de onda al interior de la frontera también está relacionada con este parámetro. El caso `rand_mode` $= 3$ es el que tiene mayor tendencia a mantener el frente esférico; los demás casos tienden a tener una velocidad superior a la velocidad del modelo debido a que $V_{max} - V_{mod} > V_{mod} - V_{min}$, especialmente en el caso `rand_mode` $= 2$. En todos los casos se observa que la energía distorsionada por la frontera aleatoria se retrasa respecto al frente de onda principal.

En los *snapshots* tomados a los 3 segundos, gran parte de la energía ha vuelto al centro del modelo. Se observa, al igual que en la figura anterior, que el frente de onda se conserva con gran

intensidad con `rand_mode` $= 2$ debido a que el rango $[V_{min}\, V_{max}] = [1600\, 9057]\, m/s$ genera muy pocos valores por debajo de la velocidad del modelo $V_{mod} = 2000 m/s$. Cuando `rand_mode` $= 1$ y `rand_mode` $= 3$ se observa un frente de onda con menos energía. En todos estos casos el frente de onda observado es el mismo frente incidente en la frontera que llegó al borde del modelo y volvió al centro del mismo. Finalmente el mejor resultado se obtiene con `rand_mode` $= 0$, donde el frente de onda incidente fue desvanecido.

A modo de conclusión sobre este parámetro, se observa que tener un rango simétrico respecto a V_{mod} con `rand_mode` $= 3$ permite eliminar la reflexión generada por el primer contacto del frente de onda con la frontera aleatoria, pero al tener un intervalo tan pequeño, no tiene la capacidad de desvanecer el frente de onda incidente que atraviesa toda la frontera. Por otro lado tener el rango más grande posible con `rand_mode` $= 0$ permitió tener el mejor resultado al difuminar el frente de onda.

Por otro lado, se experimentó el efecto del periodo empleado para cambiar la frontera aleatoria `ks_rand`. Las Figuras 23 y 24 muestran los *snapshots* del campo en 2 y 3 [s] respectivamente para diferentes valores de `ks_rand` desde 2 hasta 10000 pasos de tiempo. Los parámetros del modelo y el modelado son los mismos de los experimentos anteriores descrito por las Tablas 8 y 9; y adicionalmente se estableció `rand_mode` $= 3$. Para los *snapshots* tomados a 2 segundos, en el interior del modelo se observa que el campo tiene características similares; pero observando el campo al interior de la frontera, se observa que el proceso de inestabilidad ha comenzado para los valores más pequeños de `ks_rand`. Esta inestabilidad se manifiesta en la saturación que tienen algunos

puntos del campo y es más evidente en los *snapshots* tomados a los 3 segundos, tanto al interior de la frontera como al interior del modelo. Este comportamiento es coherente con el mostrado en la Tabla 11.

La causa de la inestabilidad está relacionada con el cambio del modelo de velocidades durante la propagación y su efecto se agudiza cuando el parámetro `rand_mode` establece un rango más grande de variación de las velocidades aleatorias. En comparación con los parámetros del experimento realizado por Fletcher and Robertsson (2011), la principal diferencia está en el método numérico empleado, debido a que los experimentos de este trabajo se emplearon diferencias finitas, mientras que Fletcher and Robertsson (2011) empleó el operador pseudodiferencial descrito por Zhang and Shen (2010).

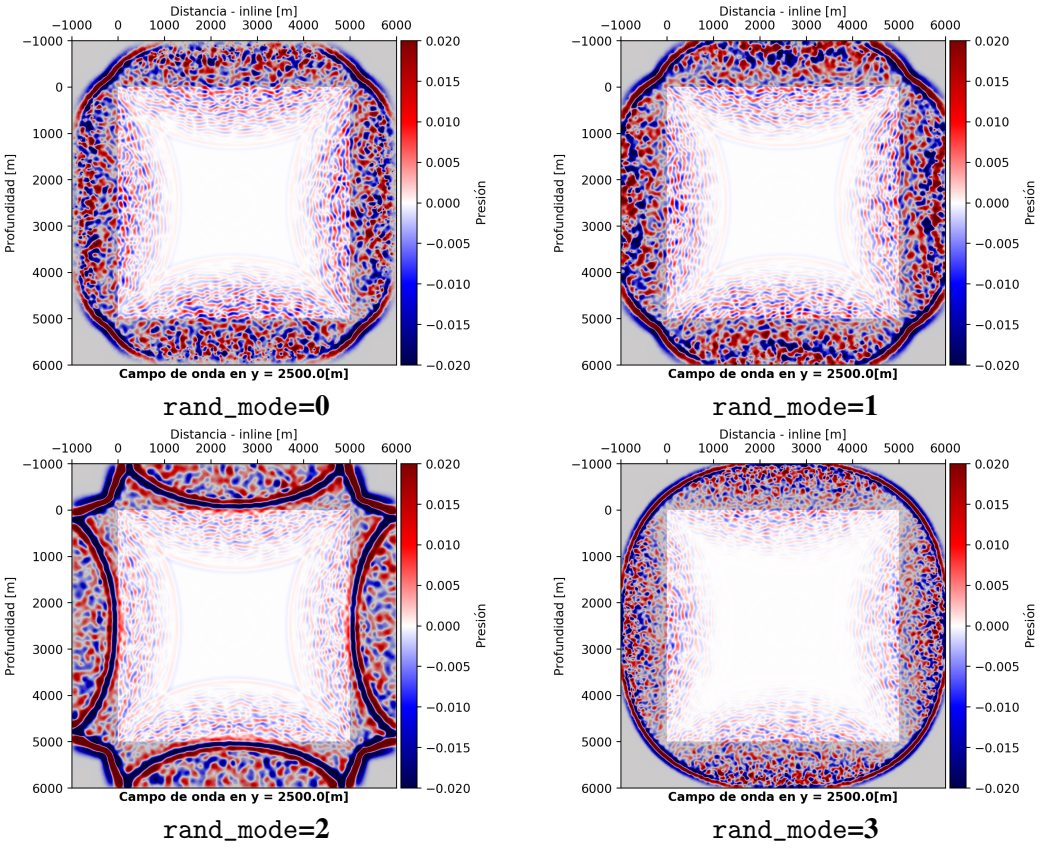


Figura 21. Comparación de diferentes valores de `rand_mode`. Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 2$ [s] y en el corte $y = 2500$ [m].

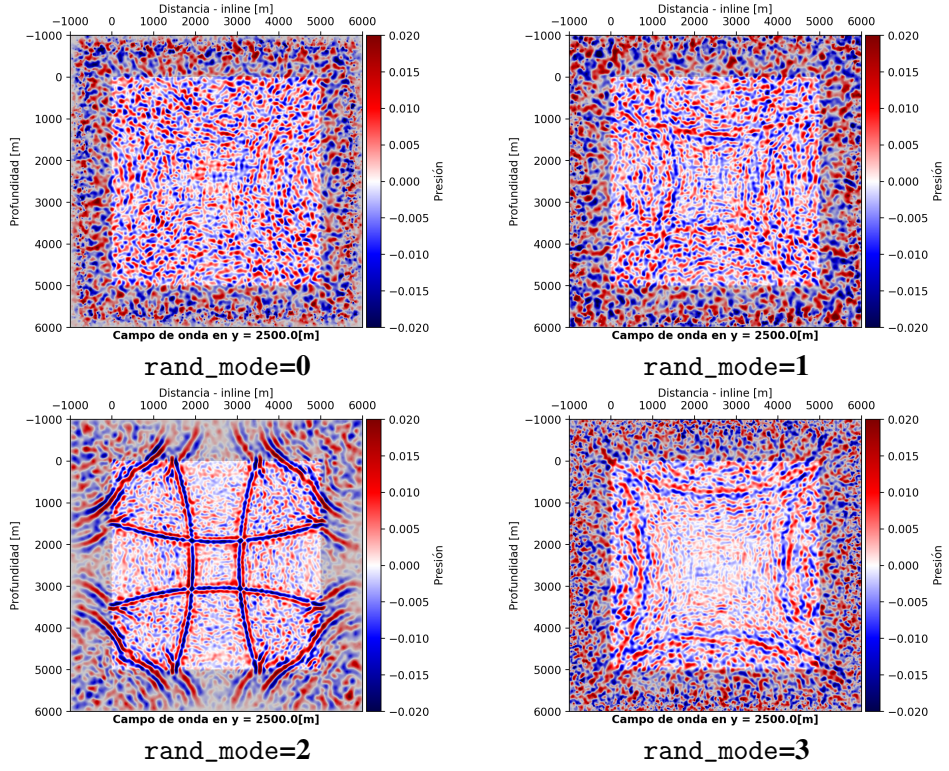


Figura 22. Comparación de diferentes valores de `rand_mode`. Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 3$ [s] y en el corte $y = 2500$ [m].

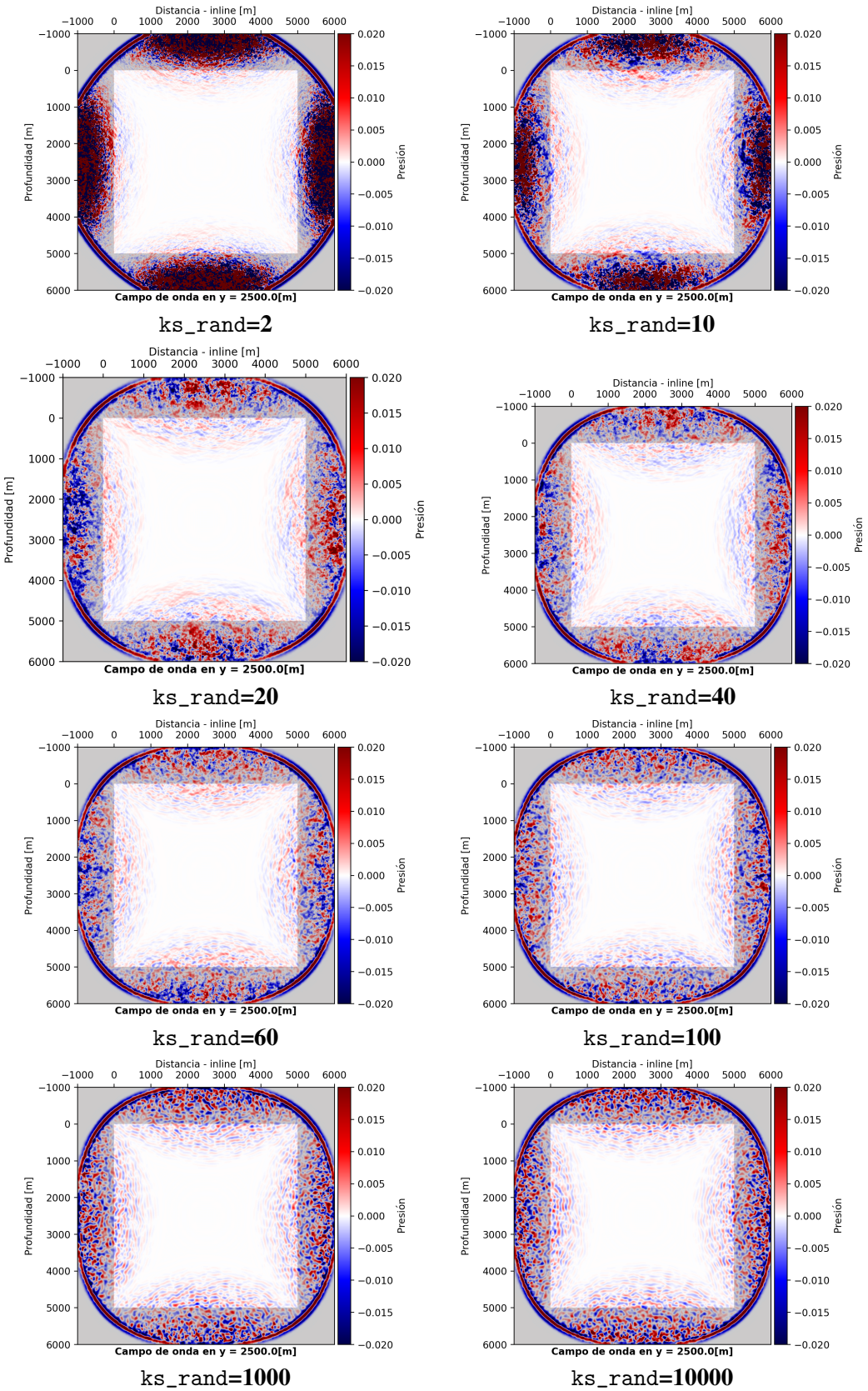


Figura 23. Comparación de diferentes valores de `ks_rand` desde 1 hasta 10000. Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 2$ [s] y en el corte $y = 2500$ [m]. En esta prueba se fijó `rand_mode` $= 3$.

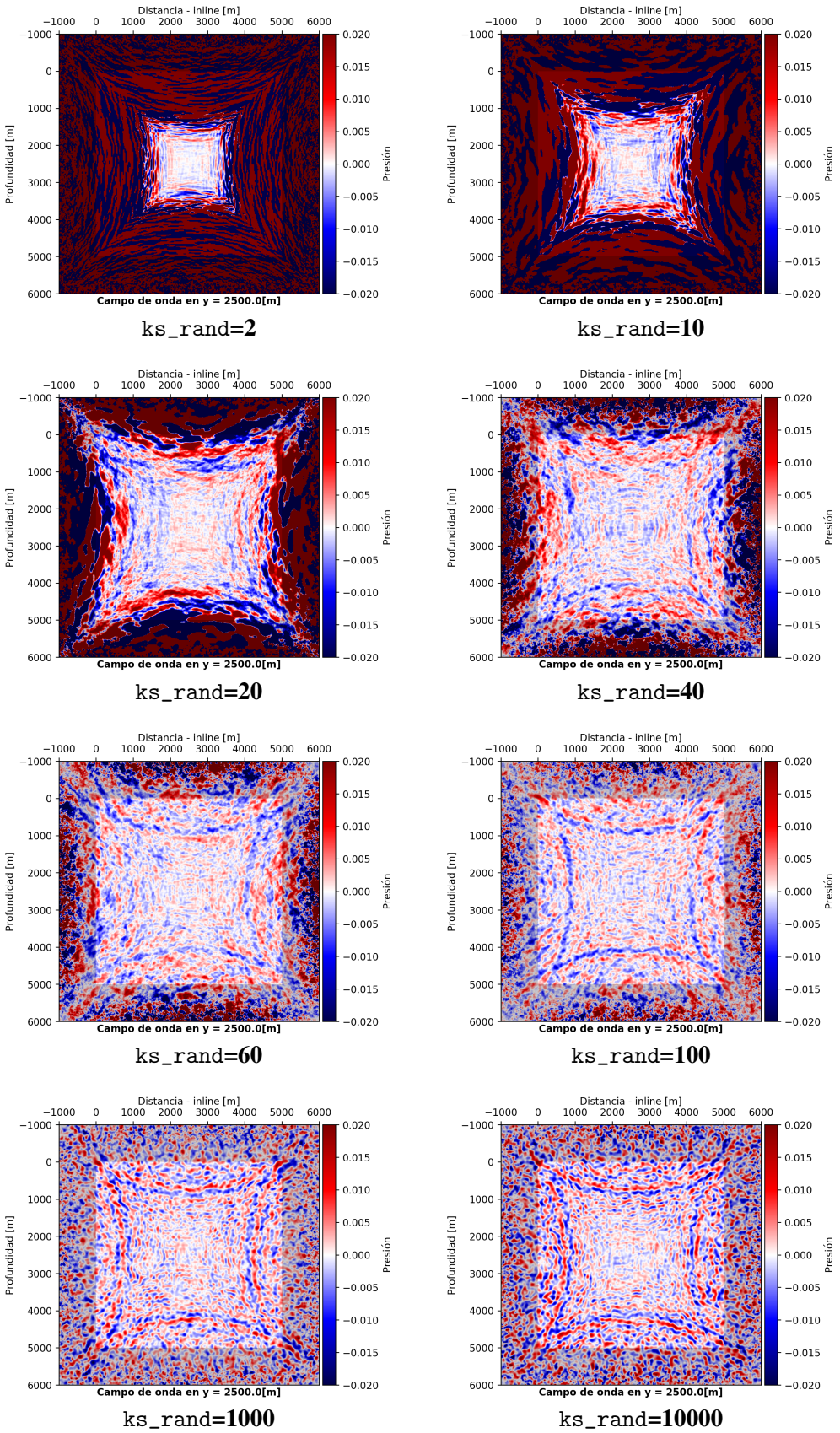


Figura 24. Comparación de diferentes valores de `ks_rand` desde 1 hasta 10000. Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 3$ [s] y en el corte $y = 2500$ [m]. En esta prueba se fijó `rand_mode` $= 3$.

2.5.3.2. Efecto de la longitud de la frontera aleatoria (L). Las Figuras 25 y 26 muestran *snapshots* del campo de la fuente para 2 y 3 segundos respectivamente con variaciones del ancho de la frontera aleatoria desde $\texttt{L} = 8$ hasta $\texttt{L} = 56$; para los cuales se mantuvieron constantes los parámetros $\texttt{ks\_rand} = 10000$ $\texttt{rand\_mode} = 3$. En las dos Figuras se evidencia la presencia de un frente de onda como resultado de la energía que atraviesa la frontera aleatoria y vuelve al modelo. Los *snapshots* tomados a los 2 segundos evidencian que entre más pequeña sea la frontera, la energía vuelve más rápido al modelo y con mayor intensidad, como era de esperarse. En este punto también se observa que la frontera permite retener temporalmente la energía que entra en ella, lo cual produce un efecto positivo adicional al de la frontera aleatoria en sí.

En la Figura 26 se observan los *snapshots* tomados a los 3 segundos. Los casos donde $L = 8$ y $L = 16$ no dan información útil debido a que se ven afectados por los frentes de onda reflejados desde las caras paralelas al plano de la visualización, los cuales llegan perpendicularmente al plano de corte. En el resto de *snapshots* se observa que el frente de onda reflejado tiene menor amplitud a medida que L se vuelve más grande, debido a que su paso por la frontera aleatoria toma más tiempo.

En general el parámetro L grande favorece el efecto de la frontera aleatoria debido a dos factores: primero, incrementa el nivel de distorsión que sufre el frente de onda en su paso por la frontera aleatoria; y segundo, retiene temporalmente en la frontera la energía no deseada.

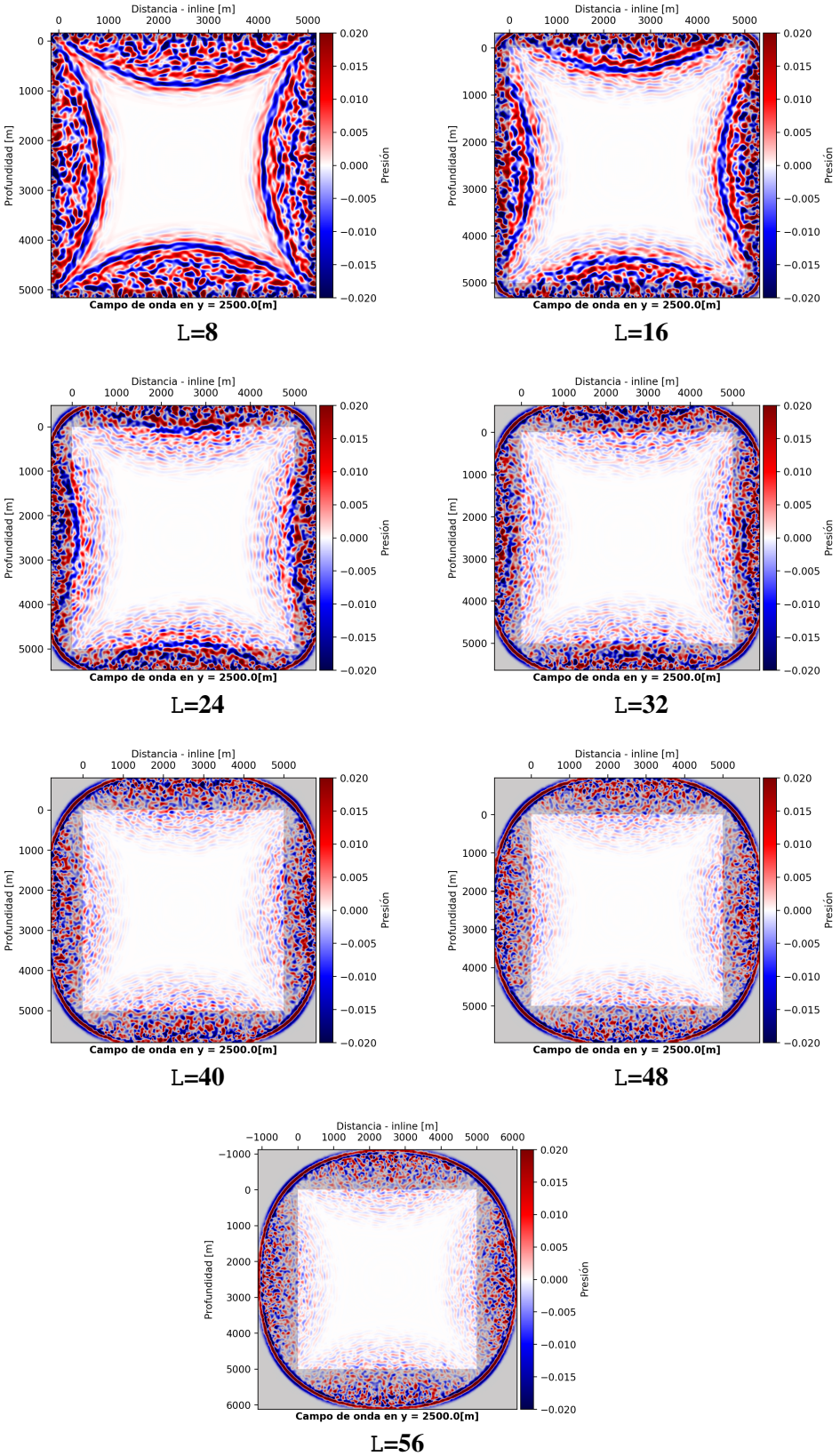


Figura 25. Comparación de diferentes valores de longitud de la frontera aleatoria (L). Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 2$ [s] y en el corte $y = 2500$ [m]. En esta prueba se fijó `rand_mode` $= 3$.

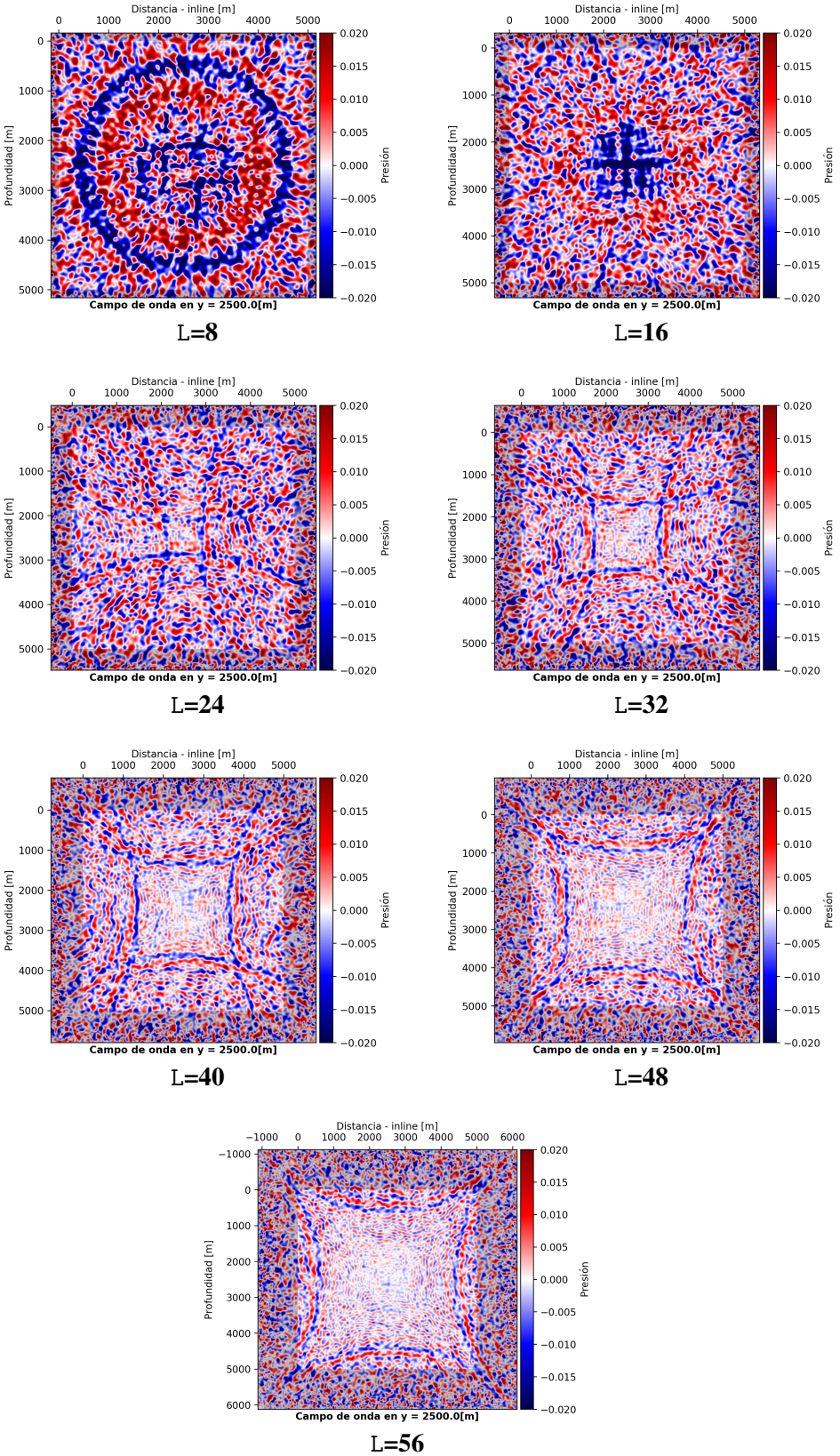


Figura 26. Comparación de diferentes valores de longitud de la frontera aleatoria (L). Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 3$ [s] y en el corte $y = 2500$ [m]. En esta prueba se fijó `rand_mode` $= 3$.

2.5.3.3. Efecto de la función de distribución de velocidades aleatorias. Por último se exploró el efecto de cambiar la función r_d que establece el rango de variación de la velocidad en la frontera aleatoria (ver Figura 4). Las funciones r_d implementadas fueron: lineal, exponencial, cuadrática y constante ($r_d = 1$). Las Figuras 27 y 28 muestran los *snapshots* del campo de la fuente para 2 y 3 segundos en cada caso.

El caso donde $r_d = 1$ evidencia la necesidad de suavizar los cambios de velocidad en la frontera aleatoria debido a que el inicio de la frontera aleatoria forma un contraste de impedancia acústica muy marcado. A los 2 segundos se observan dos frentes de onda: el frente reflejado con una fuerte amplitud y el frente transmitido que tiene una baja amplitud y ha sido distorsionado por los valores aleatorios de velocidad. A los 3 segundos se observa una gran cantidad de energía concentrada en el interior del modelo donde adicionalmente se ha superpuesto la energía reflejada en las caras paralelas al plano de corte.

Los otros casos presentan características muy similares. A los 2 segundos el frente de onda transmitido se encuentra completamente contenido en la frontera y la energía reflejada al interior del modelo corresponde a reflexiones aleatorias. A los 3 segundos los casos donde r_d es exponencial y cuadrática no presentan diferencias significativas entre ellas; por otro lado el caso lineal tiene mayor cantidad de energía en el centro del modelo. Y en los tres casos se puede observar un frente de onda que corresponde a la energía reflejada en el borde del modelo.

2.5.3.4. Resumen del efecto de los parámetros de la Estrategia 3. Después del estudio de los parámetros `rd_type`, `rand_mode`, `ks_rand` y `L`, se pudo ver que:

- Cambiar el campo de velocidades mediante el parámetro `ks_rand` puede generar inestabilidad en el esquema de diferencias finitas. Puede ser una incompatibilidad entre el parámetro y el método numérico en particular.

- El parámetro `rand_mode` $= 3$ parece ser la mejor configuración debido a que que emplea el rango más amplio de valores aleatorios y parece no verse afectado por velocidades que pueden aparecer que harían inestable el esquema de diferencias finitas.

- El ancho de la frontera aleatoria L mejora la acción de la misma. Un valor más alto de L permite que la distorsión del frente de onda se complete en un espacio mucho mayor y así se retrasa el tiempo en el que la energía del frente vuelve al interior del modelo.

- La función elegida para suavizar el cambio de velocidades entre el modelo y la frontera aleatoria (que se modifica mediante el parámetro `rd_type`) no tiene efectos notorios en los frentes de onda observados. El caso extremo $rd = 1$ hace evidente la necesidad de la función para suavizar el cambio de velocidad.

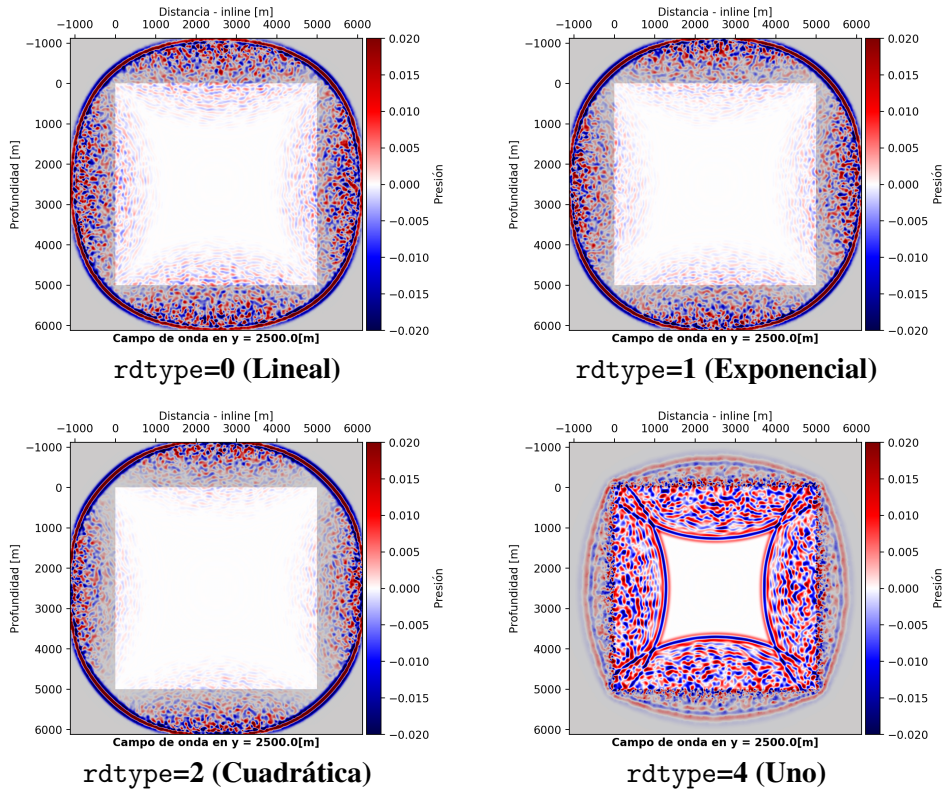


Figura 27. Comparación de diferentes valores de `rdtype`. Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 2$ [s] y en el corte $y = 2500$ [m]. Se fijó `rand_mode` $= 3$.

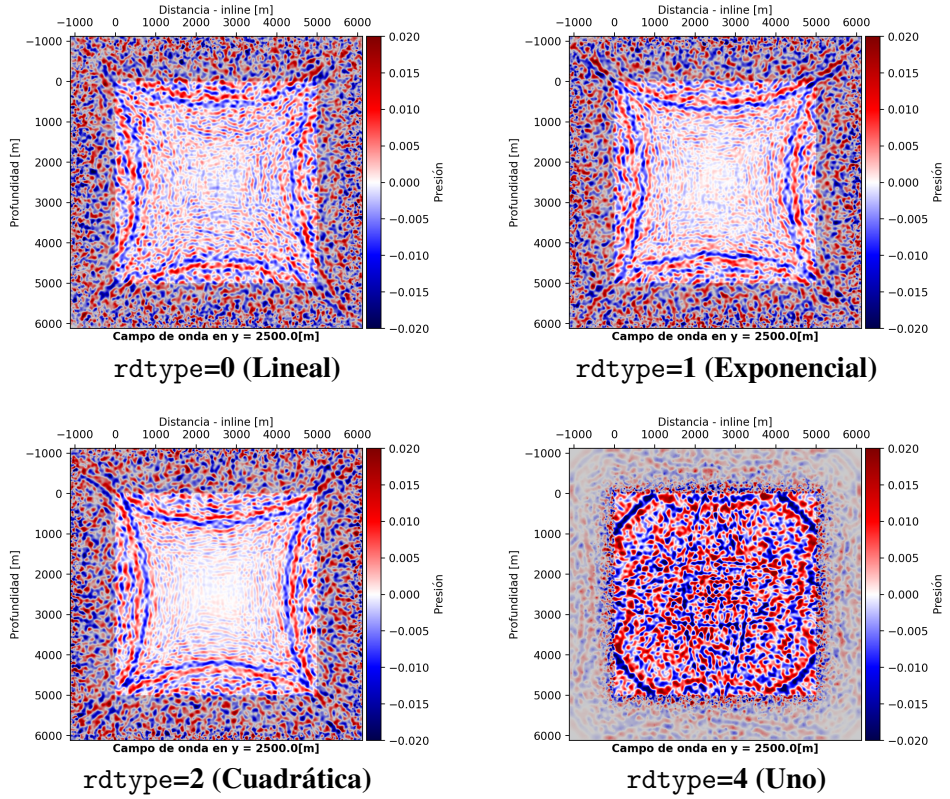


Figura 28. Comparación de diferentes valores de `rdtype`. Se muestra el *snapshot* del campo de la fuente modelado en el tiempo $t = 3$ [s] y en el corte $y = 2500$ [m]. Se fijó `rand_mode` $= 3$.

2.5.4. Requerimiento de memoria y cómputo en las diferentes estrategias. Para llevar a cabo la comparación de memoria requerida y cómputo se procesó únicamente el disparo 65 de la adquisición realizada sobre el modelo SEG-EAGE. La Tabla 12 muestra la información específica de este disparo que se ubica en el sector central del modelo.

Tabla 12
Parámetros del disparo 65 de la adquisición sobre el modelo SEG-EAGE

Posición de la fuente [m]	$(6240, 6240, 0)$
Rango de receptores en x [m]	$[4240, 8240]$
Rango de receptores en y [m]	$[4240, 8240]$
Número de trazas [trazas]	2601

Para la estrategia 1, se ejecutaron pruebas variando el valor de `ks_store` desde 48 (valor usado en la migración del dato completo) hasta 180 haciendo incrementos de 12 unidades. En cuanto a la calidad del dato de salida, el resultado fue siempre el mismo, debido a que esta estrategia siempre hace propagación hacia adelante, por lo tanto el único efecto de modificar `ks_store` fue en los tiempos de ejecución y la memoria requerida.

Las estrategias 2 y 3 se ejecutaron de la misma forma que cuando se procesó todo el dato. La Figura 29 permite comparar las estrategias mediante el tiempo de ejecución (en el eje vertical) y la memoria requerida en la GPU (en el eje horizontal). Se observa que las mejores características las tiene la estrategia 3, tanto en memoria como en cómputo; pero así mismo tuvo el mayor error, como

se mostró en la sección 1.4.3. La estrategia 2 tuvo un error menor con un tiempo de ejecución similar a la estrategia 3 pero consumiendo casi el 100 % de la memoria disponible. Otro inconveniente de esta estrategia es su falta de flexibilidad dado que no tiene un parámetro que permita ajustar la memoria requerida. Finalmente la estrategia 1 no introduce errores al hacer la reconstrucción y tiene el parámetro *ks_store* que le permite modificar sus requerimientos de memoria.

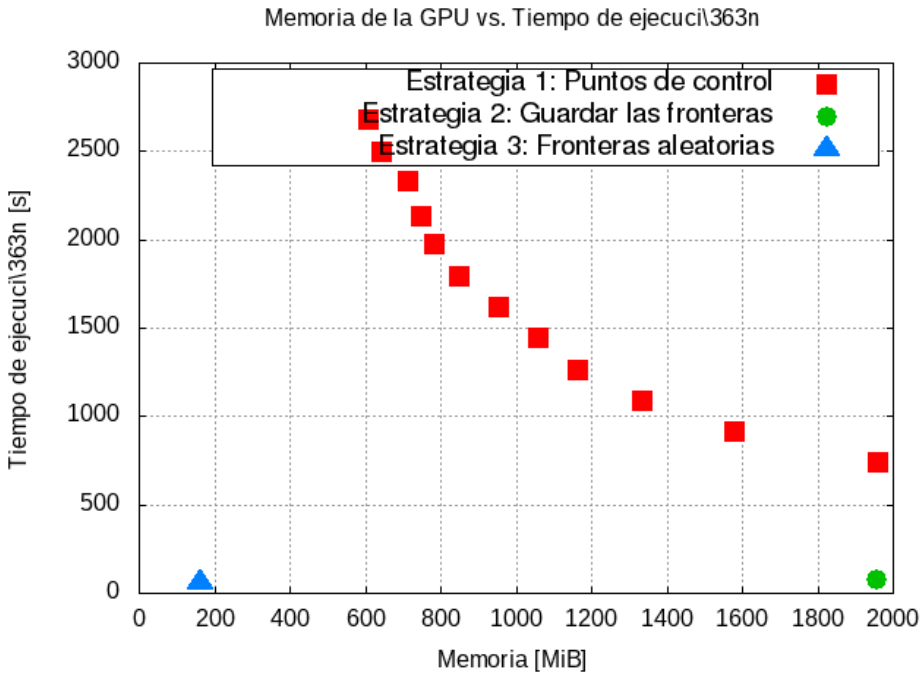


Figura 29. Comparación de las estrategias en función de la memoria y el tiempo de ejecución para migrar un solo disparo.

Tabla 13
Datos de la Figura 29 para la comparación de las estrategias en función de la memoria y el tiempo de ejecución para migrar un solo disparo.

Estrategia	`ks_store`	**Texe [s]**	**Memoria GPU [Bytes]**
1	48	736.314491	2055140736
1	60	911.046646	1655764440
1	72	1084.406150	1401615888
1	84	1261.685905	1220081208
1	96	1444.874980	1111160400
1	108	1618.419175	1002239592
1	120	1791.141345	893318784
1	132	1971.944274	820704912
1	144	2128.043569	784397976
1	156	2329.662774	748091040
1	168	2493.576996	675477168
1	180	2674.015058	639170232
2	–	78.642548	2052001913
3	–	58.196952	169550472

3. Análisis de tiempos de ejecución en arquitectura GPU

3.1. Introducción

En el desarrollo de software sobre plataformas HPC es importante medir el desempeño mediante métricas que permitan establecer el nivel de eficiencia de la implementación. El tiempo de ejecución es por naturaleza la primera métrica a tener en cuenta para ser optimizada, aunque hay otras que son importantes como el consumo de energía, el uso de memoria, la ocupación de los dispositivos de cómputo, la saturación de los buses de comunicación, etc.

En el proceso de optimización de las implementaciones es común el desarrollo de modelos matemáticos de predicción de las métricas, de esta forma es posible determinar los parámetros óptimos de ejecución del algoritmo (bajo las restricciones del modelo). Estos modelos se basan en parámetros de hardware y características de la aplicación para predecir el tiempo de ejecución.

Como parte del cumplimiento de los objetivos de la tesis, se implementó el algoritmo RTM sobre una GPU, por lo que la mayor parte del tiempo de ejecución corresponde al cómputo en la misma. Por ser la GPU un dispositivo independiente, se pudo medir el tiempo de ejecución sin sobrecostos causados por el sistema operativo, sin embargo la dinámica interna de la GPU es compleja y se convierte en un obstáculo para desarrollar un modelo de predicción del tiempo de ejecución.

La precisión, la complejidad y el tiempo de desarrollo del modelo depende de la cantidad de información de la aplicación y de los parámetros de la arquitectura. Un modelo que involucre todos los detalles de funcionamiento de la GPU puede llegar a ser excesivamente complejo, mientras que uno que solo se base en los parámetros de la aplicación será sencillo pero no permitirá optimizar ningún parámetro de hardware. La implementación del algoritmo se hizo mediante la segmentación de cada una de las estrategias en subrutinas que se ejecutan en la GPU llamadas *kernels*. En el presente trabajo se han desarrollado modelos de predicción del tiempo de ejecución de cada uno de los *kernels*, donde se tuvieron en cuenta los parámetros de cada *kernel* y la organización de los *threads* como único parámetro de hardware de interés.

Los *kernels* que fueron optimizados corresponden a los que consumen el mayor porcentaje de tiempo para cada una de las tres estrategias analizadas en el capítulo 2. La optimización de los *kernels* se llevó a cabo mediante algunas de las técnicas propuestas por el fabricante como el uso de memoria compartida y arreglos de datos que favorezcan los accesos *coalesced* de memoria global. El tamaño de bloque se optimizó mediante el modelo obtenido experimentalmente.

3.2. Análisis preliminar del algoritmo

La implementación del algoritmo RTM se realizó de forma modular favoreciendo la reutilización de las rutinas básicas entre las diferentes estrategias de manejo de memoria, las cuales, fueron finalmente trasladadas a la GPU como *kernels*. La primera versión de las rutinas se realizó de forma intuitiva y será referenciada en el documento como *naive*.

La Figura 30 presenta un diagrama simplificado de la estructura del código RTM. El bloque

azul representa la estrategia seleccionada para procesar cada disparo, mientras que el resto de secciones son ejecutadas independientemente de la estrategia seleccionada. La mayor parte del tiempo de ejecución se concentra en los *kernels* empleados en el desarrollo de la estrategia. La cantidad de veces que es llamado cada *kernel* es diferente según la estrategia seleccionada. La Tabla 14 muestra los *kernels* estudiados, la cantidad de veces que fueron ejecutados y el porcentaje del tiempo total de ejecución que corresponde a cada uno para procesar un dato de prueba con un disparo.

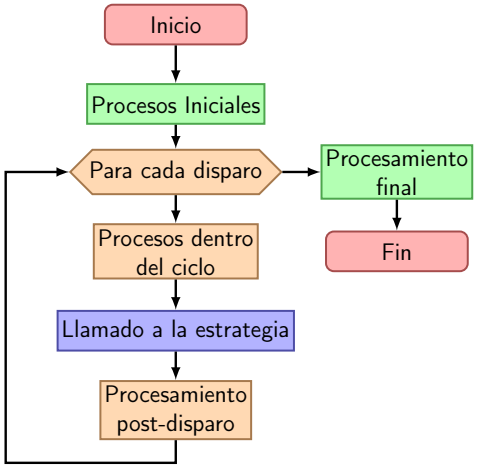


Figura 30. Estructura general de la implementación del algoritmo RTM simplificada.

Tabla 14
Lista de kernels *optimizados y su impacto sobre el tiempo de ejecución en su versión inicial.*

Kernel	**Llamados - % Tiempo de ejecución**		
	Estrategia 1	**Estrategia 2**	**Estrategia 3**
`stencil_eval_gpu`	63798 - 56.95 %	7496 - 63.94 %	7496 - 76.99 %
`apply_cpml_left_gpu`	63798 - 7.56 %	4998 - 5.72 %	2499 - 3.40 %
`apply_cpml_right_gpu`	63798 - 7.63 %	4998 - 5.79 %	2499 - 3.45 %
`apply_cpml_back_gpu`	63798 - 6.91 %	4998 - 5.22 %	2499 - 3.11 %
`apply_cpml_front_gpu`	63798 - 6.90 %	4998 - 5.21 %	2499 - 3.10 %
`apply_cpml_top_gpu`	63798 - 6.81 %	4998 - 5.19 %	2499 - 3.06 %
`apply_cpml_bottom_gpu`	63798 - 6.82 %	4998 - 5.20 %	2499 - 3.07 %
`RTM_IC_0_1`	2500 - 0.33 %	2500 - 3.13 %	2500 - 3.77 %

Algunos *kernels* no fueron incluidos dentro del procedimiento de optimización porque el porcentaje del tiempo total de ejecución es despreciable. Entre estos *kernels* se encuentran `load_boundary_*`, `save_boundary_*` empleados en la estrategia 3; y `randomize_boundary_*` empleados en la estrategia 2.

3.3. Metodología propuesta para optimizar los *kernels*

Para la optimización de los *kernels* se empleó la metodología representada en la Figura 31, la cual tiene como objetivo mejorar el tiempo de ejecución del mismo. La evaluación de desempeño incluye el desarrollo de modelos de predicción del tiempo de ejecución con base en resultados experimentales. Estos modelos mejoran la comprensión del *kernel* estudiado y favorecen una eva-

luación integral de las mejoras realizadas sobre la implementación.

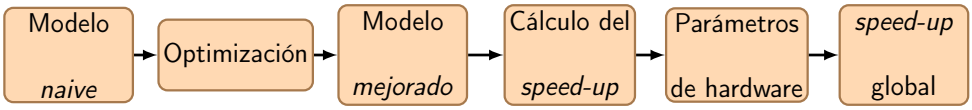


Figura 31. Metodología empleada para la optimiazión de los *kernels*.

La implementación *naive* es el punto de partida, sobre el cual se desarrolló un modelo de predicción del tiempo de ejecución. Este modelo será la referencia para evaluar el desempeño de las mejoras de la implementación. Posteriormente se implementaron algunas mejoras estándar propuestas en NVIDIA Corporation (2016) que dieron como resultado las versiones *mejoradas* de los *kernels*. El tiempo de ejecución de las versiones *mejoradas* se modeló y se comparó con la versión *naive* para obtener el *speed-up* de la mejora. Finalmente se estudió el efecto de algunos parámetros de hardware que modifican la ejecución del algoritmo y se evaluó el impacto de las mejoras en el tiempo de ejecución global del algoritmo.

3.3.1. Modelo del tiempo de ejecución *naive*. La versión *naive* de cada *kernel* fue el punto de partida para realizar la optimización del mismo. Así mismo, fue la referencia para determinar la aceleración obtenida durante este proceso. Sobre esta versión se desarrolló un modelo de predicción del tiempo de ejecución basado únicamente en los parámetros geofísicos del problema, como el tamaño del modelo, las capas de PML en cada cara del modelo y el orden de aproximación del método de diferencias finitas. Las secciones siguientes mostrarán los parámetros de estos modelos.

3.3.1.1. Kernel: `stencil_eval_gpu`**.** Este *kernel* extrapola un paso de tiempo el campo de la fuente y los receptores aplicando la solución numérica de la ecuación de onda. Este procedimiento se aplica a todo el modelo, incluyendo la zona PML. El modelo de predicción del tiempo

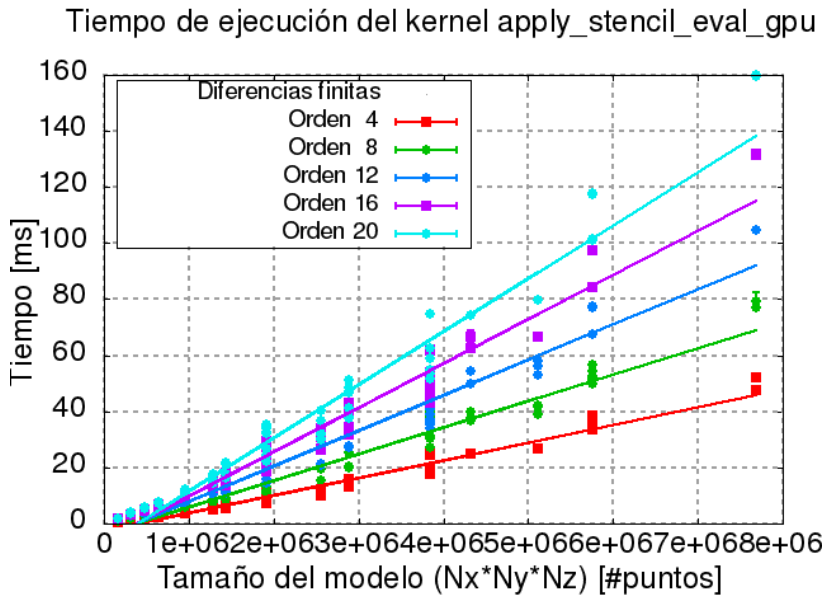
de ejecución se basó en el análisis de un conjunto de mediciones realizadas al ejecutar el *kernel* sobre datos de diferentes tamaños y empleando diferentes órdenes de aproximación en la solución de la ecuación de onda. Las ecuaciones

$$
\begin{aligned}
t_{stencil\_naive} &= m_{stencil\_naive} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}) + a_{stencil\_naive}, \\
m_{stencil\_naive} &= \beta_{1;stencil\_naive} * Orden + \beta_{2;stencil\_naive}, \\
a_{stencil\_naive} &= \beta_{3;stencil\_naive} * Orden^2 + \beta_{4;stencil\_naive} * Orden + \alpha_{stencil\_naive}
\end{aligned}
\tag{37}
$$

describen el comportamiento del tiempo de ejecución como una función lineal que depende del tamaño del campo. El intercepto y la pendiente de esta función son a su vez función del orden de aproximación de las diferencias finitas. Los datos experimentales, la curva de ajuste y los parámetros del modelo se presentan en la Figura 32. En esta figura, el eje horizontal de la gráfica corresponde al tamaño del modelo mientras que el eje vertical muestra el tiempo de ejecución. Los puntos representan experimentos individuales y los colores permiten diferenciar los experimentos según el orden de aproximación empleado. El ajuste realizado, tiene en cuenta simultáneamente el tamaño del modelo y el orden de aproximación según el conjunto de ecuaciones 37. Las líneas corresponden al ajuste realizado empleando los parámetros mostrados en la misma figura.

Para obtener este ajuste, se advertía que el orden de aproximación marcaría una tendencia en el comportamiento de los datos, por ello el primer paso del ajuste fue la agrupación de los datos según este parámetro. Posteriormente, teniendo el tamaño del modelo como único parámetro, se

ajustó cada grupo de datos mediante regresión lineal. Finalmente se observó una tendencia en la pendiente y en el intercepto de los ajustes de los grupos en función del orden de aproximación, por lo que se realizó un ajuste de estos parámetros intermedios para obtener un modelo completo que tuviera en cuenta los dos parámetros.



$\alpha_{stencil\_naive}$
-1.6791260742701

$\beta_{1;stencil\_naive}$
$7.9306140441494e-07$

$\beta_{2;stencil\_naive}$
$3.1057297725926e-06$

$\beta_{3;stencil\_naive}$
-0.003926885496416

$\beta_{4;stencil\_naive}$
-0.21497330305401

Figura 32. Ajuste del tiempo de ejecución del kernel que extrapola un paso de tiempo del campo mediante la ecuación de onda en la GPU. El kernel `stencil_eval_gpu` en su versión inicial (*naive*).

3.3.1.2. Kernels: *apply_cpml_*_gpu.* Esta familia de *kernels* suma los términos de la ecuación de onda que permiten a las regiones del borde del modelo tener propiedades absorbentes/disipativas. Estos mismos *kernels* actualizan los campos auxiliares ψ y ζ. Existe un *kernel* para cada una de las caras del espacio donde se realiza la propagación. A cada uno de estos kernels se le ha desarrollado un modelo de predicción del tiempo de ejecución como en las ecuaciones:

$$t_{cpml\_left\_naive} = \alpha_{cpml\_left\_naive} + m_{cpml\_left\_naive} * (\mathtt{L} * \mathtt{Ny} * \mathtt{Nz}) \tag{38}$$
$$m_{cpml\_left\_naive} = \beta_{1;cpml\_left\_naive} * Orden + \beta_{2;cpml\_left\_naive}$$

$$t_{cpml\_right\_naive} = \alpha_{cpml\_right\_naive} + m_{cpml\_right\_naive} * (\mathtt{L} * \mathtt{Ny} * \mathtt{Nz}) \tag{39}$$
$$m_{cpml\_right\_naive} = \beta_{1;cpml\_right\_naive} * Orden + \beta_{2;cpml\_right\_naive}$$

$$t_{cpml\_back\_naive} = \alpha_{cpml\_back\_naive} + m_{cpml\_back\_naive} * (\mathtt{Nx} * \mathtt{L} * \mathtt{Nz}) \tag{40}$$
$$m_{cpml\_back\_naive} = \beta_{1;cpml\_back\_naive} * Orden + \beta_{2;cpml\_back\_naive}$$

$$t_{cpml\_front\_naive} = \alpha_{cpml\_front\_naive} + m_{cpml\_front\_naive} * (\mathtt{Nx} * \mathtt{L} * \mathtt{Nz}) \tag{41}$$
$$m_{cpml\_front\_naive} = \beta_{1;cpml\_front\_naive} * Orden + \beta_{2;cpml\_front\_naive}$$

$$t_{cpml\_top\_naive} = \alpha_{cpml\_top\_naive} + m_{cpml\_top\_naive} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{L}) \tag{42}$$
$$m_{cpml\_top\_naive} = \beta_{1;cpml\_top\_naive} * Orden + \beta_{2;cpml\_top\_naive}$$

$$t_{cpml\_bottom\_naive} = \alpha_{cpml\_bottom\_naive} + m_{cpml\_bottom\_naive} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{L}) \tag{43}$$
$$m_{cpml\_bottom\_naive} = \beta_{1;cpml\_bottom\_naive} * Orden + \beta_{2;cpml\_bottom\_naive}$$

En todos los casos las variables independientes son el tamaño de la frontera que procesa cada uno de los *kernels* y el orden de aproximación de las diferencias finitas. Se observa que la ecuación se puede interpretar como un modelo lineal respecto al tamaño de la frontera donde la pendiente es a su vez otro modelo lineal que depende del orden de aproximación. La relación entre los datos experimentales y las curvas ajustadas se pueden observar en la Figura 33, mientras que los valores obtenidos para los parámetros de los modelos se encuentran en la Tabla 15.

Tabla 15
Valores de los parámetros de los modelos de predicción del tiempo de ejecución propuesto para los kernels *de la frontera absorbente en su primera versión (*naive*).*

Kernel	**Parámetros**		
	$\alpha_{cpml\_*\_naive}$	$\beta_{1\ cpml\_*\_naive}$	$\beta_{2\ cpml\_*\_naive}$
`apply_cpml_left_gpu`	-1.305749	5.648592e-07	5.566845e-06
`apply_cpml_right_gpu`	-1.342942	5.752421e-07	5.558013e-06
`apply_cpml_back_gpu`	-0.3523925	5.033372e-07	3.51832e-06
`apply_cpml_front_gpu`	-0.3488267	5.045264e-07	3.510769e-06
`apply_cpml_top_gpu`	-0.2373986	4.624294e-07	3.450126e-06
`apply_cpml_bottom_gpu`	-0.2306914	4.560121e-07	3.463573e-06

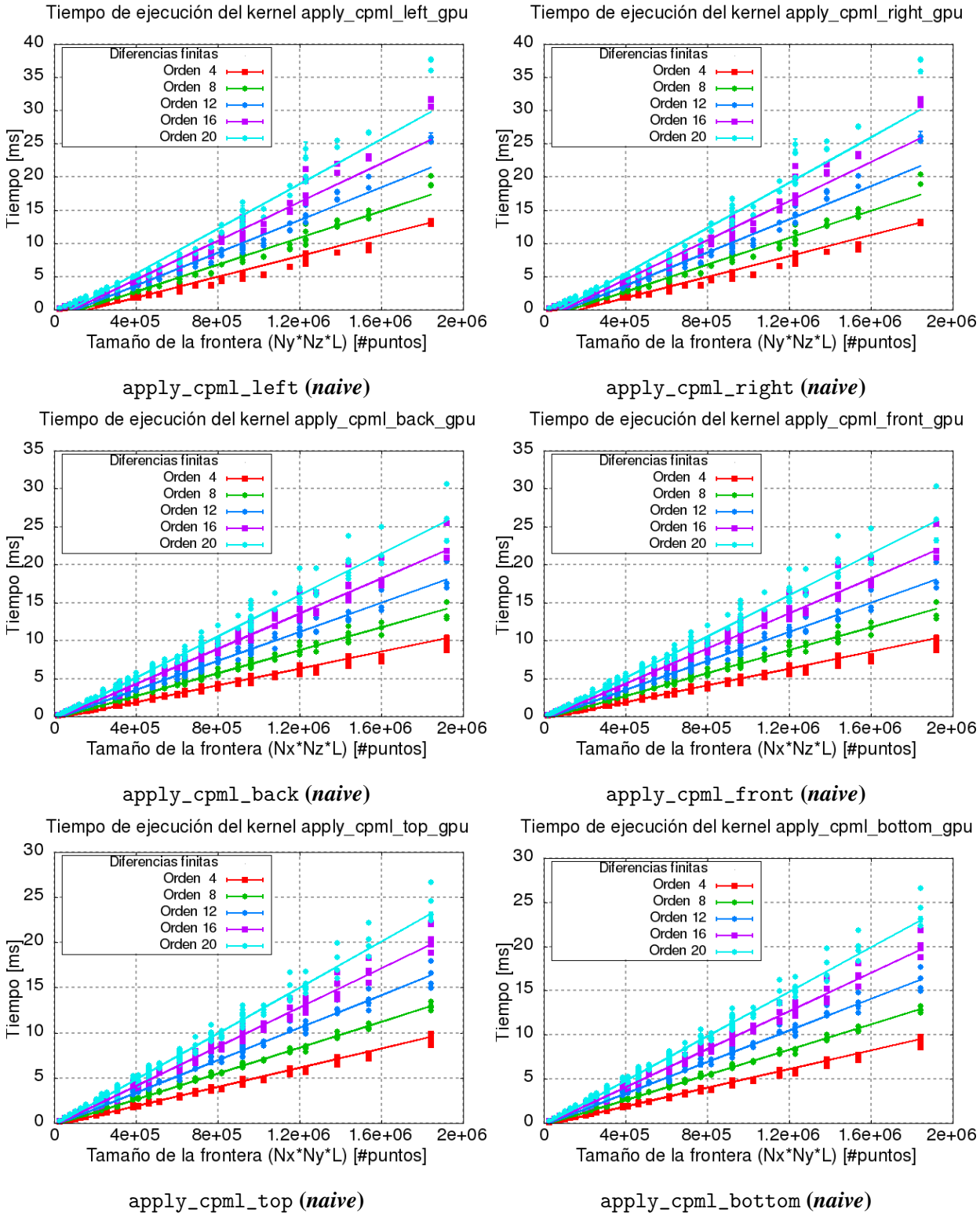
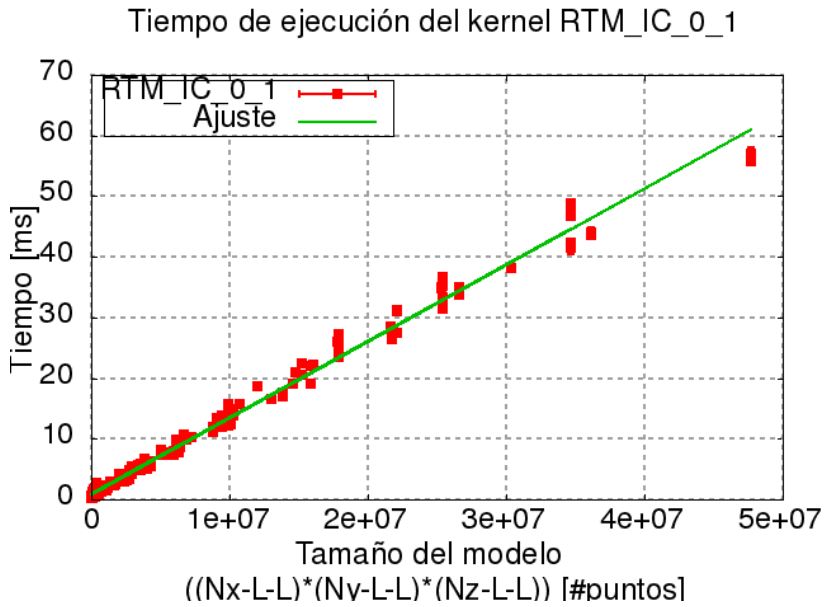


Figura 33. Ajuste del tiempo de ejecución de los *kernels* de la frontera absorbente. Para cada uno de los *kernels* se hizo un modelo de regresión lineal que tiene como variables independientes el tamaño de la frontera y el orden de aproximación de las diferencias finitas.

3.3.1.3. Kernel: *`RTM_IC_0_1`.* El *kernel* `RTM_IC_0_1` calcula la condición de imagen como el producto del campo de la fuente y el campo de los receptores en cada paso de tiempo. Este *kernel* no se ve afectado por el orden de aproximación de las diferencias finitas. Se propuso un modelo lineal como en la ecuación

$$t_{RTM\_IC\_0\_1\_naive} = \alpha_{RTM\_IC\_0\_1\_naive} + \beta_{1\ RTM\_IC\_0\_1\_naive} * (\mathtt{Nx}-2\mathtt{L})(\mathtt{Ny}-2\mathtt{L})(\mathtt{Nz}-2\mathtt{L}) \tag{44}$$

donde la variable independiente corresponde al espacio dentro del *snapshot* de los campos donde se calcula la condición de imagen. El ajuste de los parámetros α y β se observa en la Figura 34.



$\alpha_{RTM\_IC\_0\_1\_naive}$
0.83471982948163

β_1 $_{RTM\_IC\_0\_1\_naive}$
$1.2615553505727e-06$

Figura 34. Ajuste del tiempo de ejecución de la condición de imagen calculada en la GPU mediante el *kernel* `RTM_IC_0_1` en función de la cantidad de puntos procesados en su versión *naive*.

3.3.2. Optimización de los *kernels*. Los *kernels* en su versión *naive* tienen como primer objetivo obtener resultados correctos, pero no sacan el mayor provecho a la arquitectura disponible. Dentro de la metodología de diseño APOD propuesta por NVIDIA NVIDIA Corporation (2016), se busca mejorar progresivamente la versión *naive* con algunas técnicas sugeridas por ellos. En esta sección se darán detalles de algunas de estas mejoras aplicadas sobre los *kernels* modelados anteriormente y posteriormente se estimará la aceleración de la misma.

***3.3.2.1. Alineación de las estructuras de datos y la jerarquía de los* threads.** Los arreglos multidimensionales empleados en la implementación son abstracciones de arreglos unidimensionales, los cuales se ajustan mejor a la arquitectura de la memoria de un sistema de cómputo. Un macro como el siguiente

```
1 #define M(ix,iy,iz) M[Ny*Nz*(ix)+Nz*(iy)+(iz)]
```

permite indexar un espacio de memoria lineal (`M`) de tamaño `Nx*Ny*Nz` mediante tres índices (`ix`, `iy` e `iz`) como si se tratara de un arreglo de tres dimensiones. En este ejemplo un avance del índice `iz` implica un avance de una posición de memoria en el arreglo unidimensional, mientras que un incremento en `iy` e `ix` implica un incremento de `Nz` y `Nz*Ny` posiciones de memoria respectivamente. Como convención en este documento, se denominará a `z` la dimensión rápida.

Según el principio de localidad de la memoria caché, se debe procurar el acceso a datos de un mismo sector de memoria mientras se realiza el algoritmo. Esto permite sacar mejor provecho de los diferentes niveles de caché del sistema. En el caso de la GPU se busca que los *threads* de un mismo *warp* realicen accesos a posiciones consecutivas de memoria, lo que se denomina acceso *coalesced*. Este tipo de accesos favorecen significativamente el desempeño de la GPU ya que pueden ser atendidos simultáneamente por el controlador de memoria global.

Por convención, NVIDIA indexa los *threads* en tres dimensiones haciendo que la dimensión rápida sea `x`, lo cual no es coherente con la convención geofísica empleada en la implementación *naive* donde `z` es la dimensión rápida. Por ello, esta mejora consiste en alinear las estructuras de los datos y el indexado de los *threads* en los *kernels* para incrementar los accesos *coalesced* y así mejorar el uso del bus de memoria de la GPU.

Esta mejora fue aplicada a todos los *kernel* de la implementación obteniendo un alto impacto sobre el tiempo de ejecución de cada uno de ellos.

3.3.2.2. Uso de Memoria Compartida. La GPU es un gran conjunto de unidades de procesamiento en una arquitectura SIMD, las cuales se agrupan internamente en *Streaming Multiprocessors* (SM). Cada SM dispone de recursos de uso exclusivo de las unidades de procesamiento que lo componen. Entre esos recursos se encuentra la memoria compartida, la cual puede ser manipulada por el programador como una memoria caché L1 no automática. El uso de esta memoria permite reducir los accesos a la memoria global de la GPU los cuales tienen un alto costo en términos de latencia.

Los *kernels* que deben acceder varias veces a una misma posición de memoria para lograr su objetivo pueden ser adaptados para almacenar temporalmente los resultados parciales en la memoria compartida antes de volver a llevarlos a la memoria global.

El *kernel* `stencil_eval_gpu` fue modificado para hacer uso de la memoria compartida.

3.3.2.3. Uso de Memoria Constante. La memoria constante, al igual que la memoria global, es un espacio ubicado en la memoria RAM del dispositivo con una política de administración orientada a mejorar la lectura de unos pocos datos por parte de una gran cantidad de *threads*. El manejo de la caché L1 para este tipo de memoria permite reducir los accesos redundantes a memoria global.

Este tipo de memoria fue utilizado para almacenar las constantes del método de diferencias finitas en los *kernels* que calculan derivadas como: `stencil_eval_gpu` y los asociados con la aplicación de CPML `apply_cpml_*_gpu`

3.3.3. Modelo del tiempo de ejecución de las versiones mejoradas.. Con el objetivo de verificar la reducción de los tiempos de ejecución en los *kernels* mejorados, se ajustaron los modelos de las versiones *naive*, dando como resultado un nuevo modelo de predicción del tiempo de ejecución. Inicialmente se verificó si solo era necesario volver a calcular los parámetros o si era necesario plantear nuevamente un nuevo modelo. Los modelos actualizados se presentan a continuación:

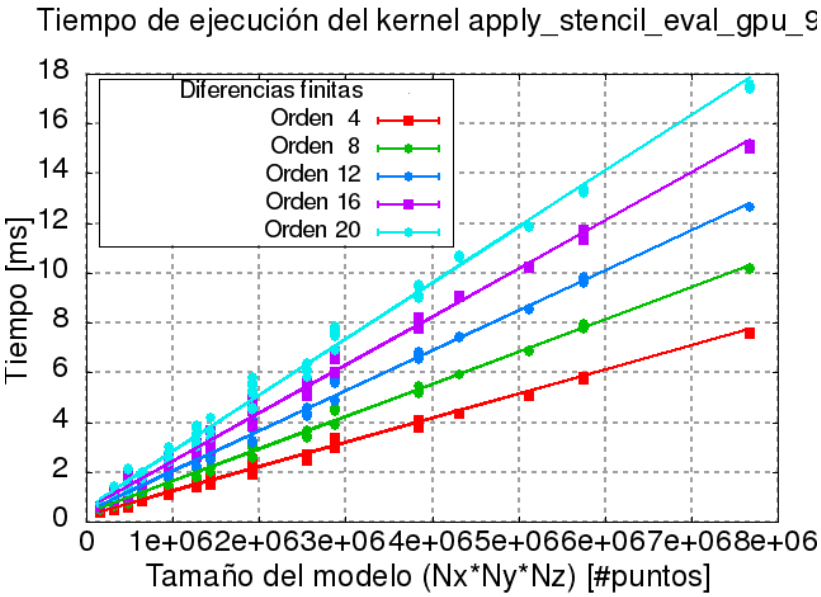
3.3.3.1. Kernel: `stencil_eval_gpu`**.** Debido al alto impacto que tiene este *kernel* sobre el tiempo de ejecución, sobre él se aplicaron todas las mejoras mencionadas anteriormente. Su tiempo de ejecución $t_{stencil\_mejorado}$ se modeló de igual forma que su forma *naive* mediante las siguientes ecuaciones

$$t_{stencil\_mejorado} = m_{stencil\_mejorado} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}) + a_{stencil\_mejorado} \tag{45}$$

$$m_{stencil\_mejorado} = \beta_{1;stencil\_mejorado} * Orden + \beta_{2;stencil\_mejorado}$$

$$a_{stencil\_mejorado} = \beta_{3;stencil\_mejorado} * Orden^2 + \beta_{4;stencil\_mejorado} * Orden + \alpha_{stencil\_mejorado}.$$

Los valores de los parámetros y los ajustes se presentan en la Figura 35.



$\alpha_{stencil\_mejorado}$
0.19064387887962

$\beta_{1;stencil\_mejorado}$
$7.975432e-08$

$\beta_{2;stencil\_mejorado}$
$6.5531810970337e-07$

$\beta_{3;stencil\_mejorado}$
$9.0578991306399e-06$

$\beta_{4;stencil\_mejorado}$
0.019265638204372

Figura 35. Ajuste del tiempo de ejecución del *kernel* que extrapola un paso de tiempo del campo mediante la ecuación de onda en la GPU. El *kernel* `stencil_eval_gpu` en su versión mejorada.

3.3.3.2. Kernels: `apply_cpml_*_gpu`. Los *kernels* relacionados con CPML en la frontera se mejoraron mediante el uso de la memoria constante y alineando la jerarquía de los *threads*. El modelo del tiempo de ejecución mantuvo la misma forma con nuevos valores para sus parámetros:

$$t_{cpml\_left\_mejorado} = \alpha_{cpml\_left\_mejorado} + m_{cpml\_left\_mejorado} * (\mathtt{L} * \mathtt{Ny} * \mathtt{Nz}) \quad (46)$$

$$m_{cpml\_left\_mejorado} = \beta_{1;cpml\_left\_mejorado} * Orden + \beta_{2;cpml\_left\_mejorado}$$

$$t_{cpml\_right\_mejorado} = \alpha_{CPML\_right\_mejorado} + m_{cpml\_right\_mejorado} * (\mathtt{L} * \mathtt{Ny} * \mathtt{Nz}) \quad (47)$$

$$m_{cpml\_right\_mejorado} = \beta_{1;cpml\_right\_mejorado} * Orden + \beta_{2;cpml\_right\_mejorado}$$

$$t_{cpml\_back\_mejorado} = \alpha_{cpml\_back\_mejorado} + m_{cpml\_back\_mejorado} * (\mathtt{Nx} * \mathtt{L} * \mathtt{Nz}) \quad (48)$$

$$m_{cpml\_back\_mejorado} = \beta_{1;cpml\_back\_mejorado} * Orden + \beta_{2;cpml\_back\_mejorado}$$

$$t_{cpml\_front\_mejorado} = \alpha_{cpml\_front\_mejorado} + m_{cpml\_front\_mejorado} * (\mathtt{Nx} * \mathtt{L} * \mathtt{Nz}) \quad (49)$$

$$m_{cpml\_front\_mejorado} = \beta_{1;cpml\_front\_mejorado} * Orden + \beta_{2;cpml\_front\_mejorado}$$

$$t_{cpml\_top\_mejorado} = \alpha_{cpml\_top\_mejorado} + m_{cpml\_top\_mejorado} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{L}) \quad (50)$$

$$m_{cpml\_top\_mejorado} = \beta_{1;cpml\_top\_mejorado} * Orden + \beta_{2;cpml\_top\_mejorado}$$

$$t_{cpml\_bottom\_mejorado} = \alpha_{cpml\_bottom\_mejorado} + m_{cpml\_bottom\_mejorado} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{L}) \quad (51)$$

$$m_{cpml\_bottom\_mejorado} = \beta_{1;cpml\_bottom\_mejorado} * Orden + \beta_{2;cpml\_bottom\_mejorado}$$

El ajuste se presenta en la Figura 36 y los parámetros en la Tabla 16.

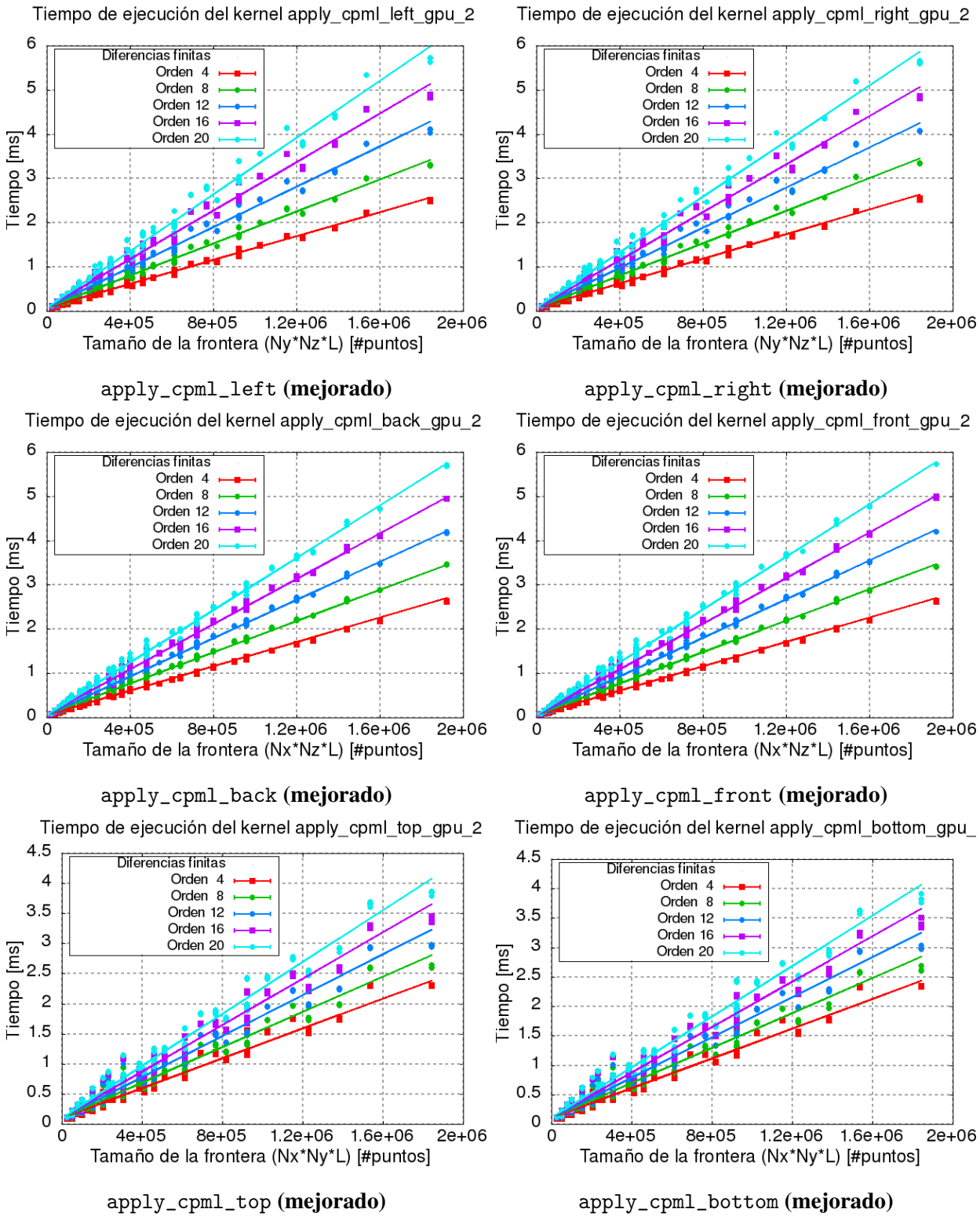


Figura 36. Ajuste del tiempo de ejecución de los *kernels* de la frontera absorbente. Para cada uno de los *kernels* se hizo un modelo de regresión lineal que tiene como variables independientes el tamaño de la frontera y el orden de aproximación de las diferencias finitas.

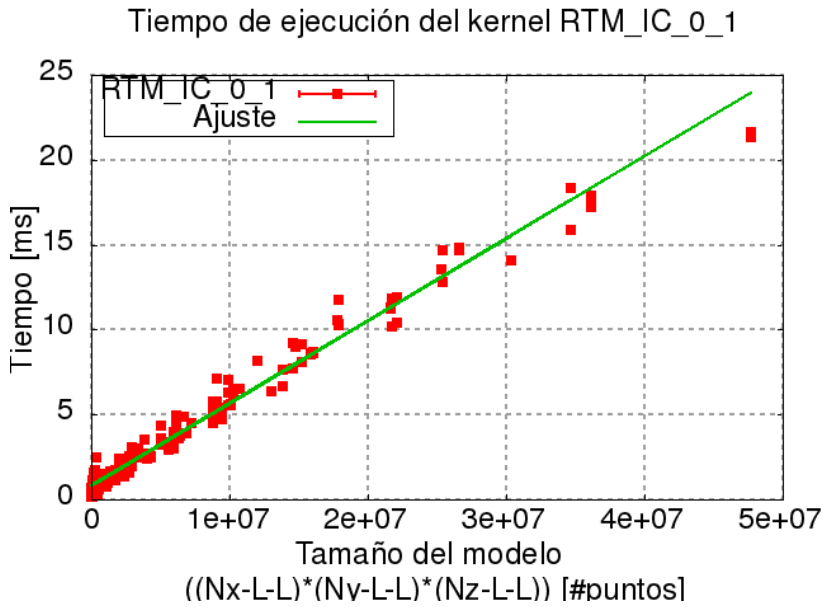
Tabla 16
Valores de los parámetros de los modelos de predicción del tiempo de ejecución propuesto para los kernels *de la frontera absorbente en su segunda versión (*mejorada*).*

Kernel	**Parámetros**		
	$\alpha_{cpml\_*\_naive}$	$\beta_{1\ cpml\_*\_naive}$	$\beta_{2\ cpml\_*\_naive}$
`apply_cpml_left_gpu`	0.07591678	1.16077e-07	8.85564e-07
`apply_cpml_right_gpu`	0.06879264	1.094305e-07	9.562157e-07
`apply_cpml_back_gpu`	0.06028437	9.90092e-08	9.764309e-07
`apply_cpml_front_gpu`	0.05701178	9.979277e-08	9.818859e-07
`apply_cpml_top_gpu`	0.1000242	5.748528e-08	1.010731e-06
`apply_cpml_bottom_gpu`	0.1047336	5.551176e-08	1.040562e-06

3.3.3.3. Kernel: *`RTM_IC_0_1`.* Por su baja complejidad, a este *kernel* solo le fue aplicada la mejora para alinear el indexado de las estructuras de datos con la jerarquía de *threads*. Tampoco fue necesario cambiar el tipo de modelo para su tiempo de ejecución, por lo que se modeló así:

$$t_{RTM\_IC\_0\_1\_mejorado} = \alpha_{RTM\_IC\_0\_1\_mejorado} + \beta_{1\ RTM\_IC\_0\_1\_mejorado} * (\mathtt{Nx}-2\mathtt{L})(\mathtt{Ny}-2\mathtt{L})(\mathtt{Nz}-2\mathtt{L}) \quad (52)$$

La Figura 37 presenta el ajuste realizado y los parámetros obtenidos en la versión mejorada.



$\alpha_{RTM\_IC\_0\_1\_mejorado}$
0.81511700736269

$\beta_{1\ RTM\_IC\_0\_1\_mejorado}$
$4.8459759602362e-07$

Figura 37. Ajuste del tiempo de ejecución de la condición de imagen calculada en la GPU ($t\_RTM\_IC\_0\_1$) mediante el *kernel* `RTM_IC_0_1` en función de la cantidad de puntos procesados en su versión mejorada.

3.3.4. Estimación de la mejora mediante los modelos. Los modelos de predicción de tiempo de ejecución generados permiten estimar el factor de aceleración de las mejoras realizadas sobre cada uno de los *kernels*, contemplando un amplio rango de valores de entrada. Esto permite comparar las dos versiones sin el sesgo que pueden dar los resultados de algunos experimentos específicos. La definición de factor de aceleración (S) relaciona el tiempo de ejecución de la versión inicial con la versión mejorada:

$$S = \frac{t_{naive}}{t_{mejorado}}. \tag{53}$$

Cada uno de estos tiempos depende de otros parámetros que hacen que el factor de aceleración cambie con los argumentos de la función. A continuación se presentan algunas observaciones que se pueden hacer con base en los modelos generados para los tiempos de ejecución para cada uno de los *kernels*.

3.3.4.1. Kernel: stencil_eval_gpu. De acuerdo con la definición de la Ecuación 53 y los modelos generados para el tiempo de ejecución antes y después de las mejoras, el factor de aceleración para el *kernel* `stencil_eval_gpu` es:

$$S_{stencil} = \frac{t_{stencil\_naive}}{t_{stencil\_mejorado}} = \frac{m_{stencil\_naive} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}) + a_{stencil\_naive}}{m_{stencil\_mejorado} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}) + a_{stencil\_mejorado}} \tag{54}$$

donde los valores de m y a dependen únicamente del orden de aproximación de las diferencias finitas. Esta expresión permite ver que manteniendo el orden constante, el factor de aceleración mejora conforme se incrementa el tamaño del modelo. La gráfica 38 permite ver el comportamiento asintótico de $S_{stencil}$ en función de $\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}$ para varios órdenes de aproximación.

El valor asintótico al que tiende la función es una cota superior de $S_{stencil}$, el cual varía para los diferentes órdenes de aproximación empleados. Esta cota está dada por:

$$S_{stencil} < \frac{m_{stencil\_naive}}{m_{stencil\_mejorado}} = \frac{\beta_{1;stencil\_naive} * Orden + \beta_{2;stencil\_naive}}{\beta_{1;stencil\_mejorado} * Orden + \beta_{2;stencil\_mejorado}}. \tag{55}$$

La Figura 39 presenta el comportamiento de esta cota en función del orden. Se observa que ésta incrementa conforme se incrementa el orden.

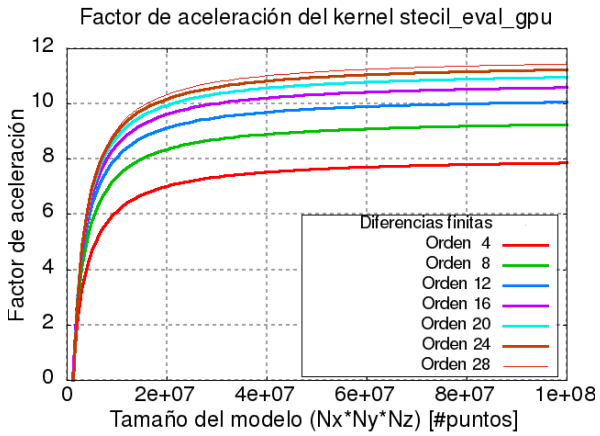


Figura 38. Factor de aceleración para diferentes órdenes de aproximación de las diferencias finitas en función del tamaño del modelo de entrada para el *kernel* `stencil_eval_gpu`. El factor de aceleración se incrementa con el tamaño del modelo y del orden de aproximación.

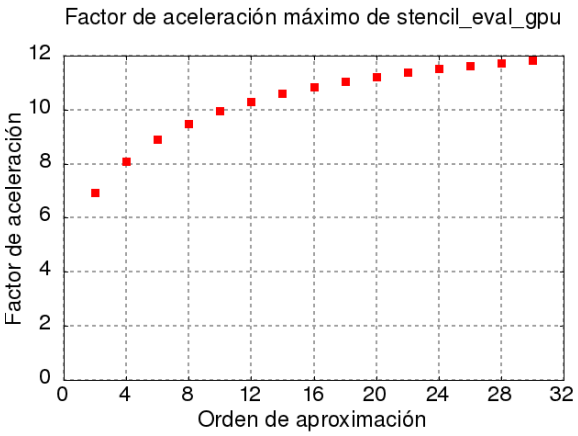


Figura 39. Factor de aceleración máximo que se puede obtener para cada uno de los órdenes de aproximación de las diferencias finitas para el *kernel* `stencil_eval_gpu`.A mayor orden, mayor expectativa de aceleración.

Finalmente es posible calcular una cota superior absoluta de aceleración que tiene en cuenta los diferentes tamaños de modelo y órdenes de aproximación. Esta cota superior absoluta representa el límite de aceleración del kernel y está dado por:

$$S_{stencil} < \frac{\beta_{1;stencil\_naive}}{\beta_{1;stencil\_mejorado}} = 13.4428 \tag{56}$$

3.3.4.2. Kernels: `apply_cpml_*_gpu`**.** Debido a la similitud de los modelos de predicción de tiempo de ejecución de estos *kernels* con el modelo del *kernel* `stencil_eval_gpu`, se observó un comportamiento similar. La Figura 40 muestra la variación del factor de aceleración según el tamaño de la frontera para diferentes órdenes de aproximación y la cota máxima de aceleración es

$$S_{left} < \frac{\beta_{1;cpml\_left\_naive}}{\beta_{1;cpml\_left\_mejorado}} = 7.21974 \qquad S_{right} < \frac{\beta_{1;cpml\_right\_naive}}{\beta_{1;cpml\_right\_mejorado}} = 7.20368$$

$$S_{back} < \frac{\beta_{1;cpml\_back\_naive}}{\beta_{1;cpml\_back\_mejorado}} = 7.99889 \qquad S_{front} < \frac{\beta_{1;cpml\_front\_naive}}{\beta_{1;cpml\_front\_mejorado}} = 7.91066$$

$$S_{top} < \frac{\beta_{1;cpml\_top\_naive}}{\beta_{1;cpml\_top\_mejorado}} = 11.863 \qquad S_{bottom} < \frac{\beta_{1;cpml\_bottom\_naive}}{\beta_{1;cpml\_bottom\_mejorado}} = 12.0733.$$

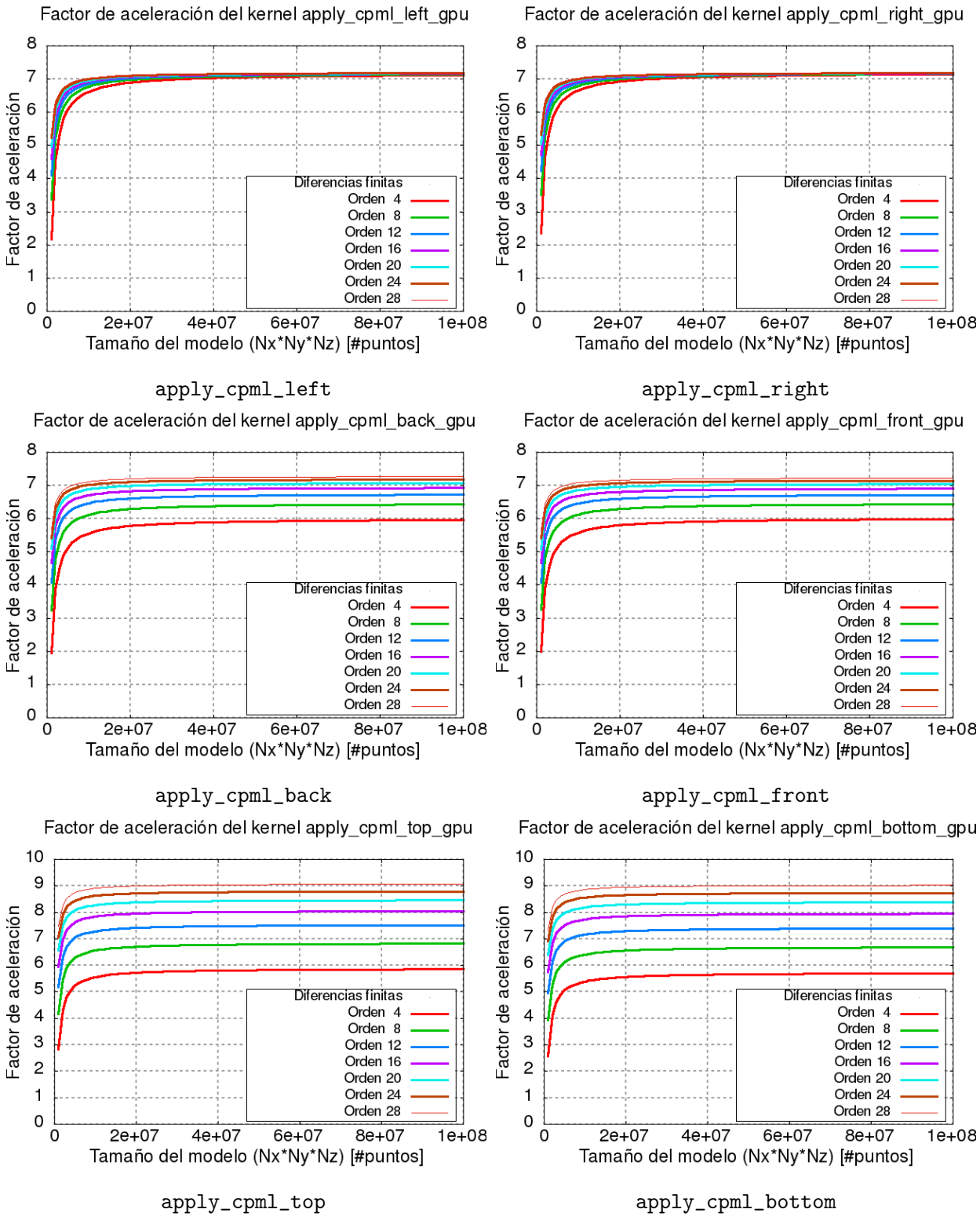


Figura 40. Factor de aceleración de los *kernels* de la frontera absorbente en función del tamaño de la frontera.

3.3.4.3. Kernel: `RTM_IC_0_1`. Este *kernel* es independiente del orden de aproximación, por lo que su análisis es más sencillo. El factor de aceleración solo depende del tamaño del modelo, el cual está dado por:

$$S_{RTM\_IC\_0\_1} = \frac{t_{RTM\_IC\_0\_1\_naive}}{t_{RTM\_IC\_0\_1\_mejorado}} = \frac{\beta_{1\ RTM\_IC\_0\_1\_naive} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}) + \alpha_{RTM\_IC\_0\_1\_naive}}{\beta_{1\ RTM\_IC\_0\_1\_mejorado} * (\mathtt{Nx} * \mathtt{Ny} * \mathtt{Nz}) + \alpha_{RTM\_IC\_0\_1\_mejorado}} \quad (57)$$

La Figura 41 presenta el comportamiento del factor de aceleración conforme aumenta el tamaño del modelo y su factor de aceleración máximo es:

$$S_{RTM\_IC\_0\_1} < \frac{\beta_{1\ RTM\_IC\_0\_1\_naive}}{\beta_{1\ RTM\_IC\_0\_1\_mejorado}} = 2.60331 \quad (58)$$

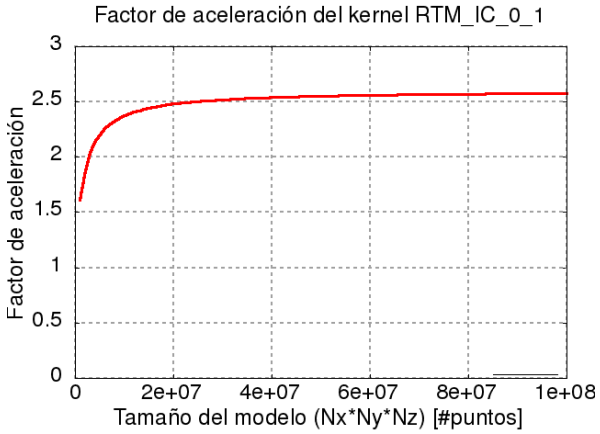


Figura 41. Factor de aceleración en función del tamaño del modelo de entrada para el *kernel* `RTM_IC_0_1`. El factor de aceleración se incrementa con el tamaño del modelo.

3.3.5. Influencia de los parámetros de hardware. Los parámetros de hardware modifican el funcionamiento de la GPU y pueden ser ajustados durante la ejecución del programa o previo a la misma.

Algunos de estos parámetros son: el tamaño del bloque de *threads*, la distribución de memoria caché L1 y la memoria compartida, etc. Estos parámetros se han mantenido constantes en los modelos desarrollados, los cuales permiten estimar el tiempo de ejecución variando únicamente parámetros geofísicos del proceso como por ejemplo: el tamaño del modelo, el grosor de las fronteras absorbentes y el orden de aproximación de las diferencias finitas. En esta sección se estudió el efecto del tamaño del bloque sobre el tiempo de ejecución, para establecer una configuración adecuada para procesar un dato de entrada.

Para el análisis, se exploró experimentalmente el espacio de posibles valores que éste puede tomar y luego se consideró la posibilidad de generar un modelo matemático del mismo o complementar el modelo que se tiene. El principal objetivo de esta etapa fue encontrar la mejor configuración de los parámetros de hardware para la ejecución de los *kernels* mejorados. En el caso de que no fuera posible encontrar la mejor configuración, se determinaron restricciones sobre los parámetros de hardware que garanticen un tiempo de ejecución cercano al mínimo. A continuación se presentan los resultados obtenidos al estudiar cada uno de los *kernel* mejorados.

3.3.5.1. Kernel: stencil_eval_gpu. Para estudiar este *kernel* se realizaron experimentos manteniendo constante las variables independientes del modelo desarrollado previamente, es decir, se mantuvo constante el orden de aproximación $Orden = 10$ y el tamaño del modelo $\{\texttt{Nx}, \texttt{Ny}, \texttt{Nz}\} = \{200, 128, 200\}$. Se exploró variando el tamaño del bloque $\{\texttt{bsx}, \texttt{bsy}, \texttt{bsz}\}$ desde 1 hasta 32 en cada una de sus dimensiones. Como resultado se obtuvo un conjunto de tiempos de ejecución cuyo histograma se muestra en la Figura 42, donde se observa que la mayor parte de los experimentos se concentran junto del valor mínimo.

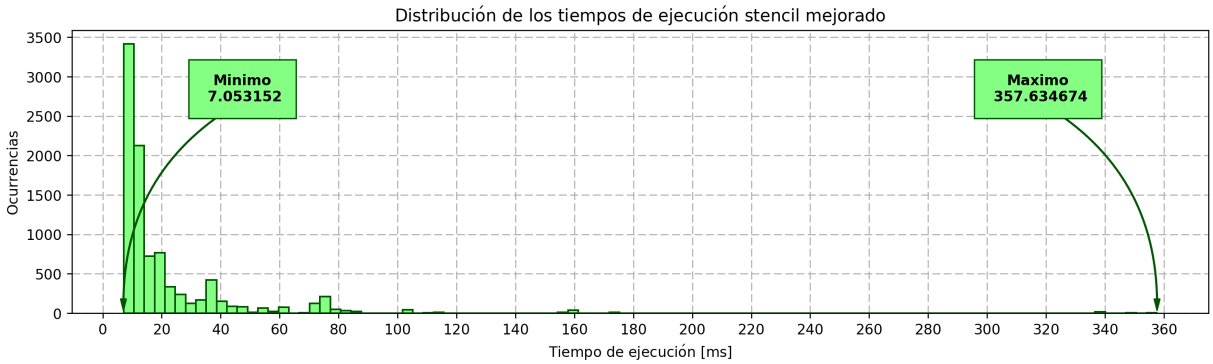
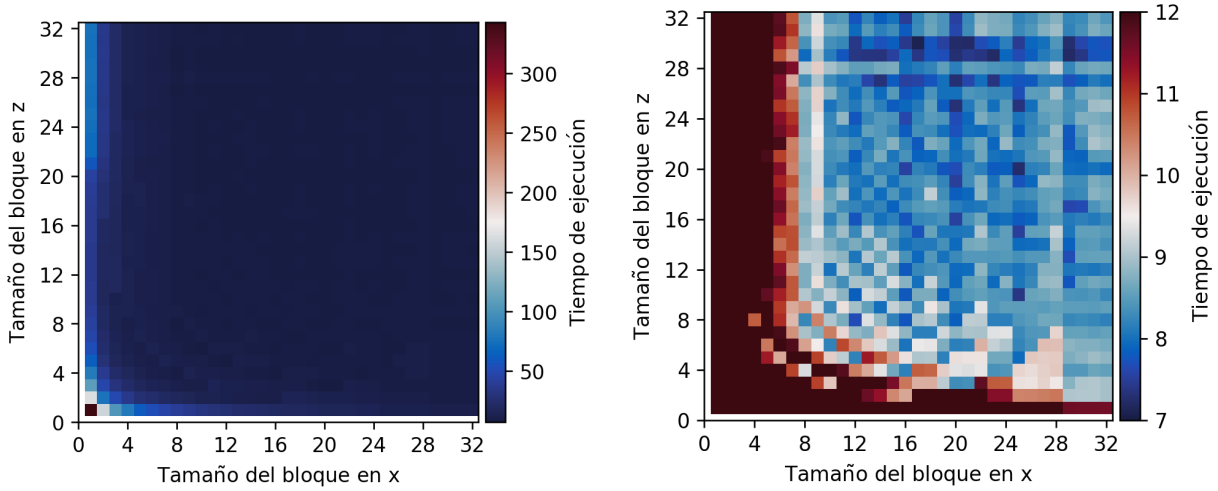


Figura 42. Histograma de experimentos obtenidos al ejecutar el *kernel* `stencil_eval_gpu` variando el tamaño del bloque de *threads* de 0 a 32 en cada una de las dimensiones. En total se registraron 9424 experimentos.

Otra observación realizada sobre los datos tiene que ver con el bajo impacto que tiene `bsy` sobre el tiempo de ejecución, lo cual es coherente con el modo en que se programó la versión mejorada de este *kernel* donde se asignó a un solo *thread* el cálculo de todos los puntos en la dirección y. Esto permitió simplificar el análisis de los resultados al reducir la dimensionalidad del espacio de análisis de $\{\texttt{bsx}, \texttt{bsy}, \texttt{bsz}\}$ a $\{\texttt{bsx}, \texttt{bsz}\}$.

La Figura 43 muestra el tiempo de ejecución en función de $\{\texttt{bsx}, \texttt{bsz}\}$. El tiempo de ejecución es el resultado de promediar los experimentos que comparten el mismo `bsy`. Inicialmente en la Figura 43(a) se observa que el valor máximo del tiempo de ejecución se encuentra cuando $\{\texttt{bsx}, \texttt{bsz}\} = \{1, 1\}$ lo cual es coherente dado el mal uso que se hace, esta configuración, de la memoria compartida. Para visualizar mejor los resultados, la Figura 43(b) muestra la misma información que la Figura 43(a) reduciendo el rango de visualización para observar un mejor contraste.

Se observó un comportamiento del tiempo de ejecución difícil de modelar con precisión y sin un



(a) Rango completo **(b) Rango restringido**

Figura 43. Variación del tamaño del bloque de *threads* $\{\texttt{bsx}, \texttt{bsz}\}$ para el *kernel* `stencil_eval_gpu`. La variable independiente `bsy` se redujo mediante el promedio debido a su bajo impacto sobre el tiempo de ejecución. La figura de la izquierda maneja todo el rango de valores desde 0 hasta 350 mientras que a la figura de la derecha se le redujo su rango para mejorar el contraste en los valores de interés.

valor mínimo claramente definido debido a la aleatoriedad que representa la experimentación misma. Por ello se decidió no modelar matemáticamente el efecto del tamaño del bloque de *threads* y por el contrario definir un conjunto de valores para los cuales se garantice un resultado cercano al mínimo. Por ejemplo, si se restringe $\texttt{bsx} > 16$ y $\texttt{bsz} > 16$, teniendo en cuenta los resultados de la Figura 43(b) se garantiza que el tiempo de ejecución no será superior a 9.173[ms]. La Figura 44 contrasta el histograma inicial con el histograma de los experimentos que cumplieron la restricción.

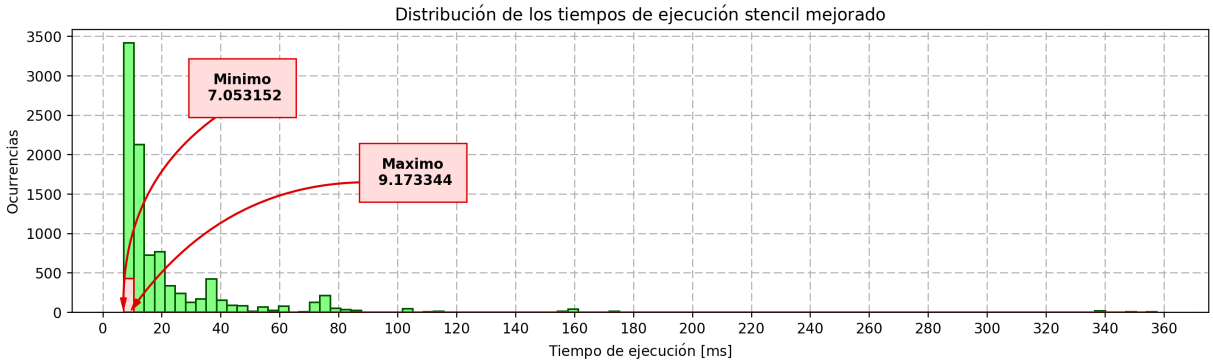


Figura 44. Contraste del histograma de los experimentos del *kernel* `stencil_eval_gpu` antes y después de restringir los valores de `bsx` y `bsz` para garantizar un valor cercano al mínimo.

En conclusión, no se modeló el tiempo de ejecución en función del tamaño del bloque sino que se propuso un conjunto de valores $\{\texttt{bsx}, \texttt{bsz}\}$ que aseguran un tiempo de ejecución reducido. En cuanto al valor de `bsy` se concluyó que no tiene un gran impacto sobre el tiempo de ejecución, pero se recomendaría usar $\texttt{bsy} = 1$ para no habilitar *threads* que no se usarán.

3.3.5.2. Kernels de la frontera absorbente. Los *kernels* relacionados con la frontera absorbente tienen una distribución similar al del *kernel* `stencil_eval_gpu`, por lo que se empleará la misma estrategia de restringir el valor del tamaño del bloque para establecer una región donde el tiempo de ejecución se conserve en niveles mínimos. Se observó que el efecto del tamaño del bloque es muy similar en las caras opuestas del cubo por lo que se analizaron en conjunto los *kernels* relacionados. A continuación se presentan los resultados del análisis hecho sobre estos *kernels*.

3.3.5.2.1. Kernels: `apply_cpml_{left,right}_gpu` La Figura 45 presentan los histogramas de los experimentos realizados recorriendo de 1 a 32 el tamaño del bloque en las direcciones *x*, *y* y *z* para los *kernels* `apply_cpml_left_gpu` y `apply_cpml_right_gpu`. Al igual que en el caso anterior, se observa una acumulación de los valores medidos cerca al valor mínimo; pero en este caso las tres variables tienen influencia

(a) ***kernel*** `apply_cpml_left`

(b) ***kernel*** `apply_cpml_right`

Figura 45. Histograma del tiempo de ejecución de los *kernels* `apply_cpml_left` y `apply_cpml_right` al variar el tamaño del bloque de *threads*. El experimento contó con 9424 ejecuciones del *kernel* con diferentes tamaños de bloque.

sobre el tiempo de ejecución, lo cual dificulta la representación de esta dependencia. Para comprender la influencia de cada parámetro de entrada sobre el tiempo de ejecución se desarrolló el siguiente procedimiento:

1. Establecer un percentil para identificar los experimentos que se considerarían con un valor aceptable. En este caso se empleó el percentil 5. La Figura 45 muestra la ubicación de esta muestra que será referenciada como umbral.

2. Para cada una de las variables independientes estudiadas se crearon subconjuntos de experimentos que comparten un mismo valor de la variable en estudio. Posteriormente se calculó el porcentaje de experimentos dentro de cada subconjunto que están por debajo del umbral. La Figura 46 presenta este

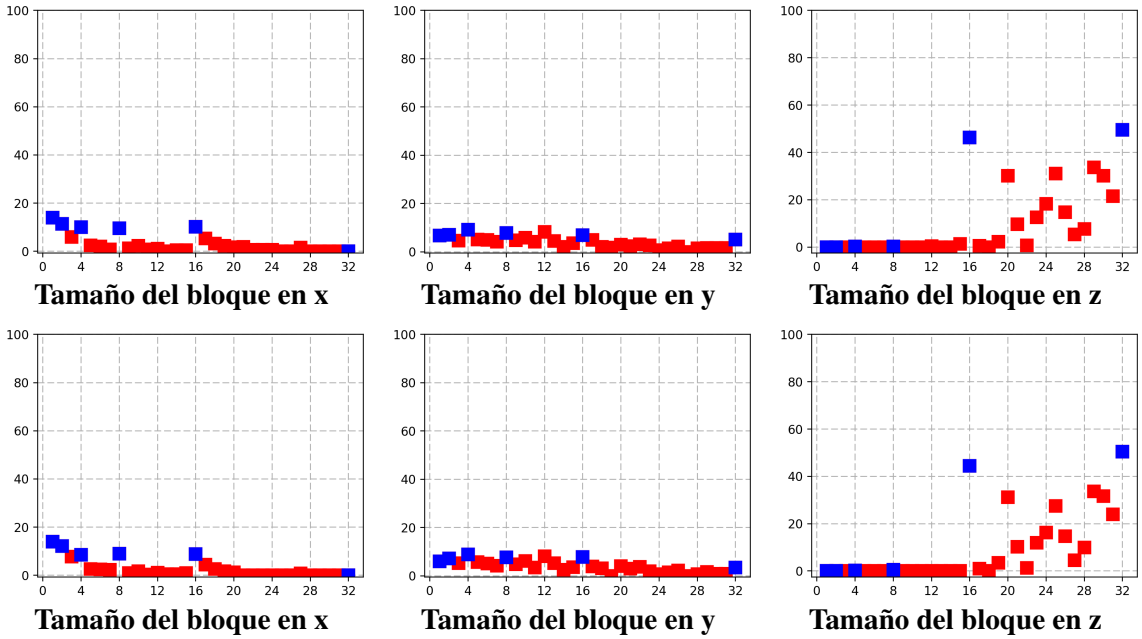


Figura 46. Porcentaje de ejecuciones que están por debajo del umbral restringiendo el tamaño del bloque en cada una de las dimensiones. Se destacan en color azul los resultados obtenidos para tamaños de bloque que son potencia de dos.

valor para cada una de las variables de estudio: `bsx`, `bsy` y `bsz`.

3. Con las gráficas obtenidas se determina un conjunto de restricciones sobre los parámetros de estudio y se analiza el conjunto de valores que cumple esta condición nuevamente en el histograma. Si se obtuvo un buen resultado con la restricción propuesta, se modifica la restricción o se establece un nuevo umbral en el numeral uno.

La Figura 46 muestra las gráficas obtenidas en el paso dos para los *kernels* en estudio. Se ha resaltado de color azul los valores donde el parámetro es igual a una potencia de dos. Se observa que siempre estos valores se destacan sobre los valores circundantes lo cual es coherente porque gran parte de los parámetros de funcionamiento de la GPU son potencias de dos, por ejemplo, la longitud de los *warps*, los segmentos de la memoria caché, la memoria manipulada en una transferencia de memoria, etc. Todo esto hace que

seleccionar tamaños de bloque con longitudes potencia de dos favorezca la coordinación de los *threads* con el hardware de la GPU.

Continuando el análisis de la Figura 46 se observa que el parámetro `bsz` es el más influyente sobre el tiempo de ejecución. Por ejemplo cuando `bsz<16` las probabilidades de encontrar un resultado bajo el umbral son casi nulas. En las gráficas de `bsx` y `bsy` no se pueden sacar conclusiones tan contundentes respecto a cómo se deberían restringir estos parámetros. Para estos *kernels* se establecieron las siguientes restricciones:

- `bsz=16` ó `bsz=32`
- `bsx` ≥ 4
- $4 \leq$ `bsy` ≤ 16

Esto redujo los valores de los parámetros de entrada de 9424 posibles combinaciones a 90. La Figura 47 muestra la distribución de los experimentos que cumplieron las restricciones y se observa una reducción significativa del rango de valores del tiempo de ejecución, además estas restricciones han logrado incluir al mínimo absoluto de esta prueba.

(a) ***kernel*** `apply_cpml_left`

(b) ***kernel*** `apply_cpml_right`

Figura 47. Contraste entre el histograma del conjunto completo de experimentos realizados variando el tamaño del bloque de *threads* para `apply_cpml_left` y `apply_cpml_right` y el conjunto restringido de los mismos. Mientras el conjunto completo consta de 9424 experimentos, el conjunto restringido consta de 90 posibles configuraciones.

3.3.5.2.2. Kernels: `apply_cpml_{back,front}_gpu` Sobre esta pareja de *kernels* se empleó el mismo procedimiento que para los *kernels* anteriores. Inicialmente en la Figura 48 se puede observar la distribución del tiempo de ejecución de los experimentos realizados. El valor de los parámetros $\{\texttt{bsx}, \texttt{bsy}, \texttt{bsz}\}$ varió de 1 a 32. La distribución es muy similar al caso anterior debido a que en estos *kernels* la variable más rápida del indexado (`ibz`) recorre de extremo a extremo el cubo.

La Figura 49 fue obtenida aplicando el punto dos del procedimiento. Se observa nuevamente que `ibz` es la variable con mayor influencia, por lo que se aplicó la misma restricción que en los *kernels* anteriores:

`ibz` $= 16$ ó `ibz` $= 32$. El comportamiento de `ibx` e `iby` se intercambia en este caso respecto a las caras `left` y `right`, por lo que las restricciones fueron intercambiadas igualmente. Las restricciones para estos *kernels* son:

- `bsz=16` ó `bsz=32`
- `bsy` ≥ 4
- $4 \leq$ `bsx` ≤ 16

Se consideraron aceptables las restricciones propuestas, lo cual se reflejó en el histograma de la Figura 50. Aunque se puede apreciar que el valor mínimo no está incluido, la reducción del rango de tiempos de ejecución es significativa.

(a) *kernel* `apply_cpml_back`

(b) *kernel* `apply_cpml_front`

Figura 48. Histograma del tiempo de ejecución de los *kernels* `apply_cpml_back` y `apply_cpml_front` al variar el tamaño del bloque de *threads*. El experimento contó con 9424 ejecuciones del *kernel* con diferentes tamaños de bloque.

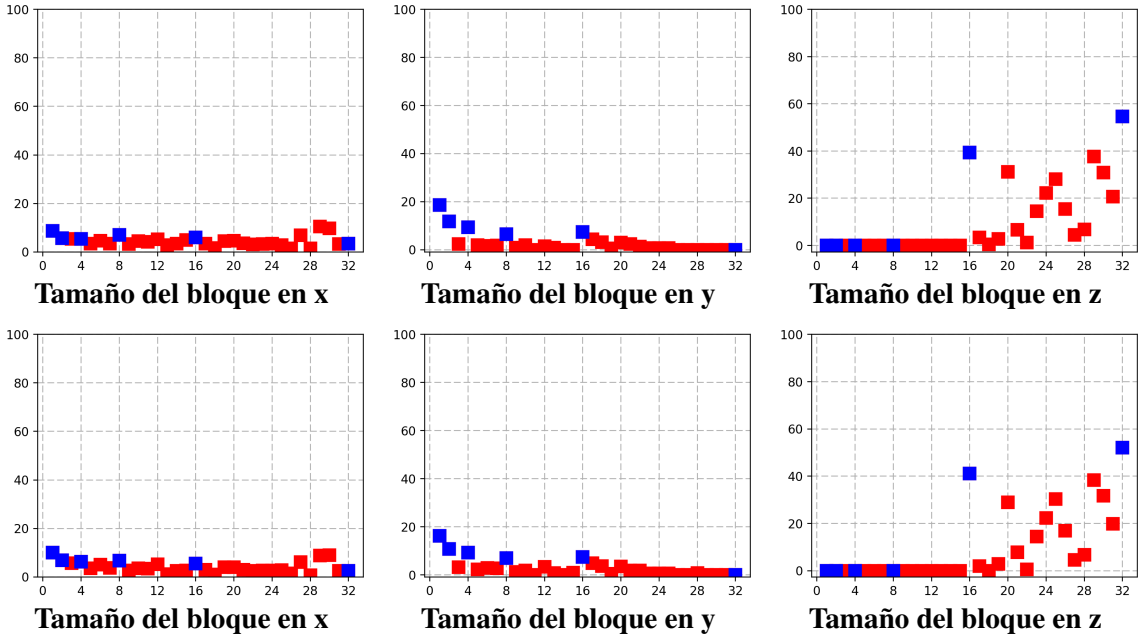


Figura 49. Porcentaje de ejecuciones que están por debajo del umbral restringiendo el tamaño del bloque en cada una de las dimensiones. Se destacan en color azul los resultados obtenidos para tamaños de bloque que son potencia de dos.

(a) ***kernel*** `apply_cpml_back`

(b) ***kernel*** `apply_cpml_front`

Figura 50. Contraste entre el histograma del conjunto completo de experimentos realizados variando el tamaño del bloque de *threads* para `apply_cpml_back` y `apply_cpml_front` y el conjunto restringido de los mismos. Mientras el conjunto completo consta de 9424 experimentos, el conjunto restringido consta de 90 posibles configuraciones.

3.3.5.2.3. Kernels: `apply_cpml_{top,bottom}_gpu` Los *kernels* de las fronteras absorbentes `top` y `bottom` tienen comportamientos diferentes a los de las demás caras del cubo. Las primeras observaciones se pueden hacer en la Figura 51, donde se aprecia que la cantidad de ocurrencias en cada intervalo es ligeramente menor a los histogramas presentados anteriormente, lo cual indica que los valores están menos concentrados en una misma región. Este fenómeno se presenta aún cuando se redujo el rango de valores de tiempo de ejecución, donde ahora el máximo no supera los $17[ms]$.

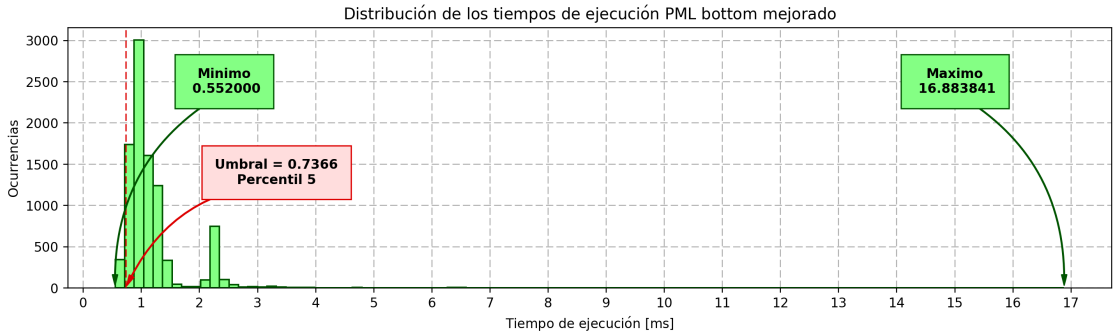
Al aplicar el procedimiento propuesto, se generaron las gráficas del segundo paso en la Figura 52. El

fenómeno más relevante en esta gráfica es la alta probabilidad de encontrar un valor por debajo del quinto percentil al restringir simplemente `ibz` $= 16$. El histograma de este subconjunto se contrasta con el conjunto completo de datos en la Figura 53. Las variables `ibx` e `iby` no tienen una marcada tendencia a favorecer o desfavorecer el tiempo de ejecución, excepto cuando estas variables toman el valor de 1 donde tienen una afectación bastante negativa. Las restricciones a los parámetros de entrada finalmente son:

- `bsz=16`
- `bsy` ≥ 2
- `bsx` ≤ 2

La Figura 54 presenta el histograma obtenido al aplicar estas restricciones a las variables de entrada del experimento. Se observa que se han eliminado solo algunos casos respecto al histograma anterior y esto redujo el rango de valores sobre el cual se distribuyen los experimentos.

(a) *kernel* `apply_cpml_top`



(b) *kernel* `apply_cpml_bottom`

Figura 51. Histograma del tiempo de ejecución de los *kernels* `apply_cpml_top` y `apply_cpml_bottom` al variar el tamaño del bloque de *threads*. El experimento contó con 9424 ejecuciones del *kernel* con diferentes tamaños de bloque.

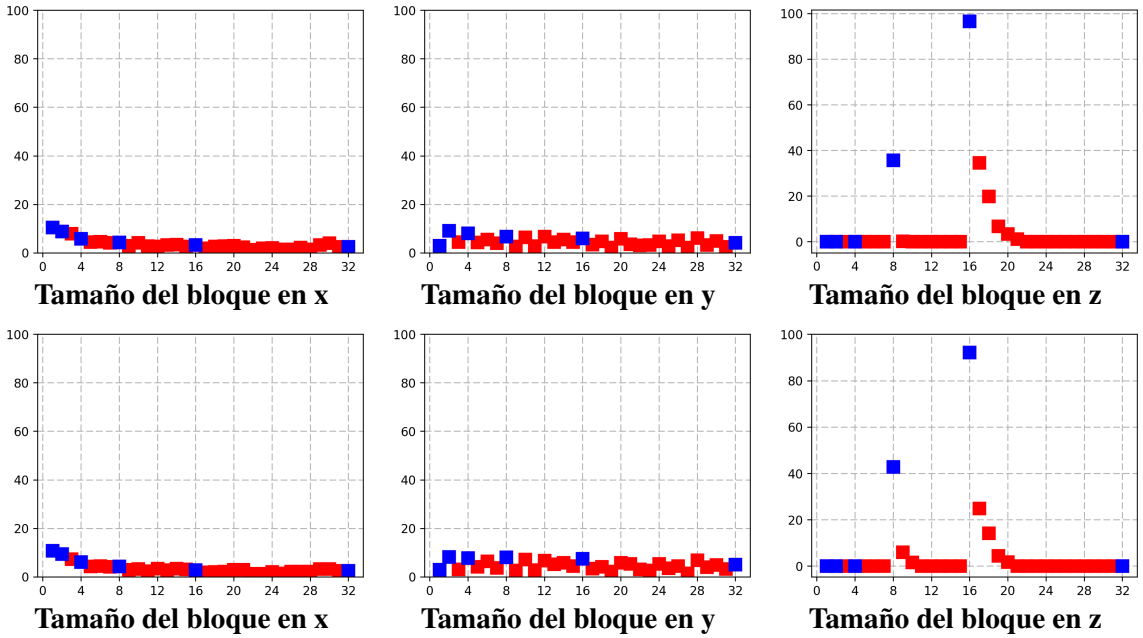
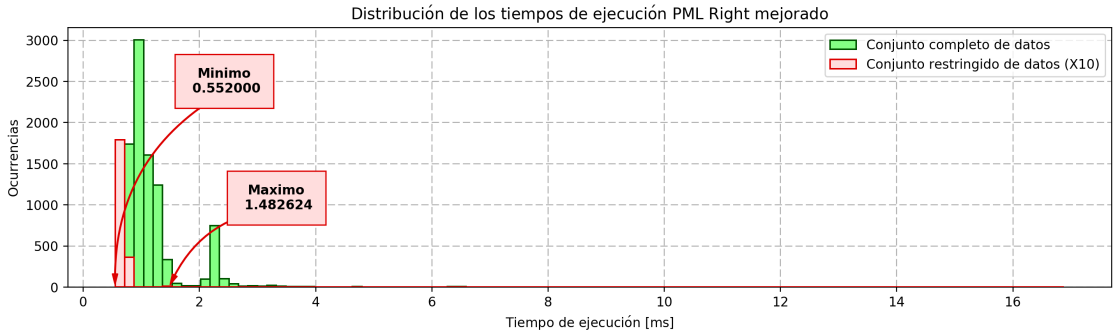


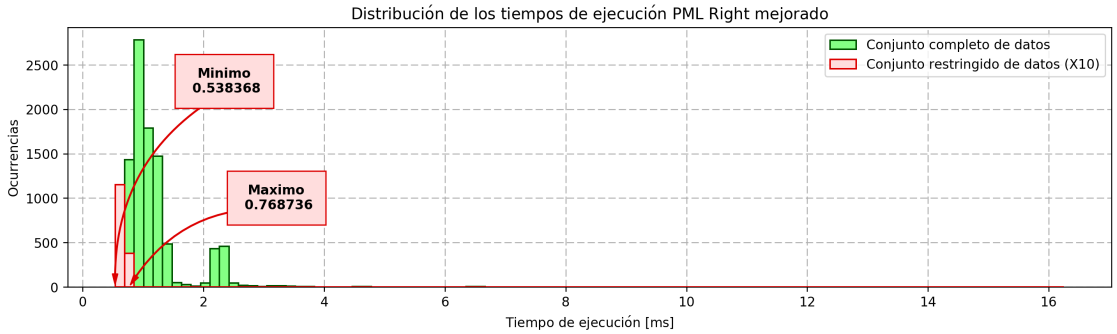
Figura 52. Porcentaje de ejecuciones que están por debajo del umbral restringiendo el tamaño del bloque en cada una de las dimensiones. Se destacan en color azul los resultados obtenidos para tamaños de bloque que son potencia de dos.

(a) ***kernel*** `apply_cpml_top`



(b) ***kernel*** `apply_cpml_bottom`

Figura 53. Contraste entre el histograma del conjunto completo de experimentos realizados variando el tamaño del bloque de *threads* para `apply_cpml_top` y `apply_cpml_bottom` y el conjunto restringido únicamente con $\texttt{bsz} = 16$.



(a) ***kernel*** `apply_cpml_top`

(b) ***kernel*** `apply_cpml_bottom`

Figura 54. Contraste entre el histograma del conjunto completo de experimentos realizados variando el tamaño del bloque de *threads* para `apply_cpml_top` y `apply_cpml_bottom` y el conjunto restringido sobre todas las dimensiones

3.3.5.3. Kernel: *`RTM_IC_0_1`.* Al igual que el *kernel* `stencil_eval_gpu`, este *kernel* actúa sobre todo el volumen de datos. Por esta razón no se afecta particularmente ninguna de las dimensiones del bloque de *threads*. Lo único que diferencia el impacto de las variables `ibx`, `iby` e `ibz` son los aspectos relacionados con el funcionamiento interno de la GPU. La Figura 55 presenta el histograma de los experimentos realizados variando el tamaño del bloque de *threads*.

Figura 55. Histograma del tiempo de ejecución del *kernel* `RTM_IC_0_1` al variar el tamaño del bloque de *threads*. El experimento contó con 20010 ejecuciones del *kernel* con diferentes tamaños de bloque.

Se aplicó el mismo procedimiento aplicado sobre todos los *kernels* de la frontera absorbente, por lo que se calculó el quinto percentil y se generó la gráfica de la Figura 56. Nuevamente se observa la gran influencia de `ibz` y se observan grandes regiones donde el valor de esta métrica es casi nulo, por ejemplo en $\texttt{iby} > 64$ ó en $\texttt{ibx} > 24$. Con base en esto se tomó como restricción

- $\texttt{bsz} \geq 100$

tras lo cual se generó el histograma de este subconjunto de datos restringidos y se comparó con el histograma original como se ve en la Figura 57. Los resultados obtenidos permiten observar una disminución en los tiempos de ejecución, por lo que se acepta como restricción definitiva.

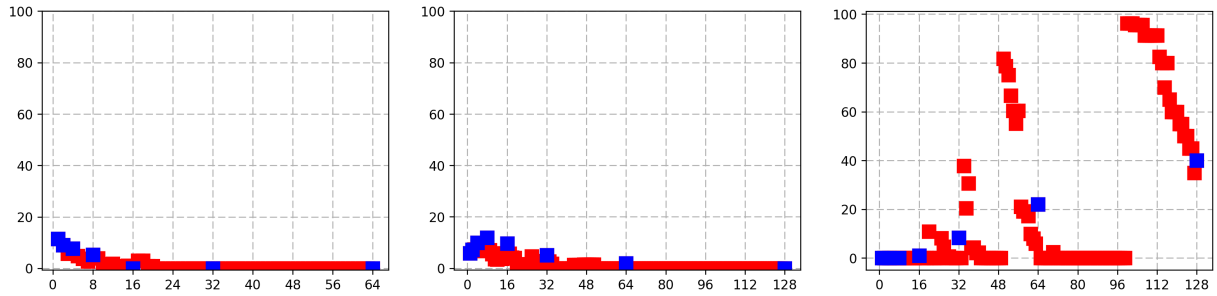


Figura 56. Porcentaje de ejecuciones que están por debajo del umbral restringiendo el tamaño del bloque en cada una de las dimensiones. Se destacan en color azul los resultados obtenidos para tamaños de bloque que son potencia de dos.

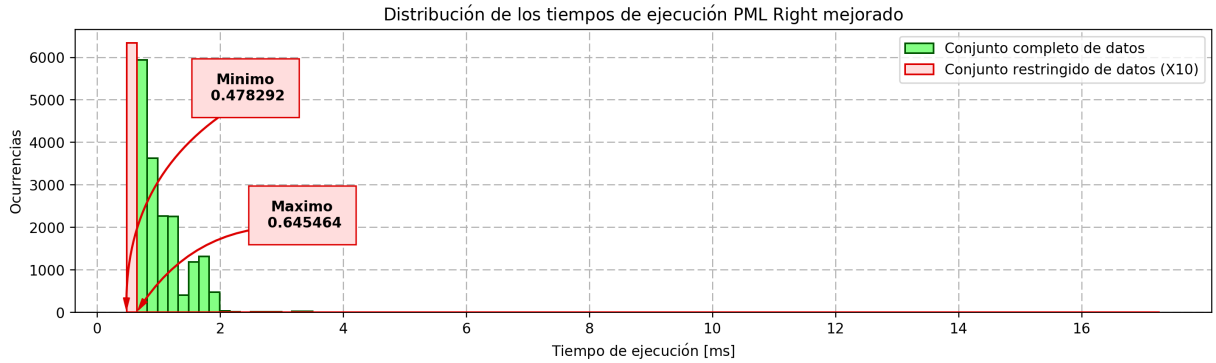


Figura 57. Contraste entre el histograma del conjunto completo de experimentos realizados variando el tamaño del bloque de *threads* para `RTM_IC_0_1` y el conjunto restringido únicamente con `bsz` ≥ 100.

3.4. Evaluación de aceleración global.

En esta sección se buscó conjugar los resultados obtenidos para cada uno de los *kernels* mejorados con el objetivo de estimar su impacto en el tiempo de ejecución global del algoritmo. Esta etapa no hace parte formal de la metodología propuesta pero apoya a un objetivo implícito del trabajo realizado en este capítulo y constituye una prueba experimental muy interesante al cuantificar la diferencia entre los resultados experimentales y los obtenidos con los modelos de predicción.

3.4.1. Medición para cada estrategia.. Con las versiones mejoradas de los *kernel* es posible medir el impacto de las mejoras sobre el tiempo global. Para ello se tomó el dato de prueba con el que se realizó el análisis preliminar del algoritmo reportado en la Tabla 14. Este dato se procesó con las versiones *naive* y las versiones mejoradas de los *kernels* en cada una de las estrategias.

En la Tabla 17 se presenta el tiempo de ejecución de la estrategia 1. Se observa que el efecto más relevante se obtiene al mejorar el *kernel* `stencil_eval_gpu`, lo cual es coherente con la información obtenida en el análisis preliminar dado que este *kernel* representa más de la mitad del tiempo de ejecución. Mejorar únicamente los *kernels* `apply_cpml_*_gpu` implica una aceleración global menor debido a que la mejora sobre estos *kernels* fue menos efectiva que la mejora sobre el *kernel* `stencil_eval_gpu`. Finalmente el impacto de mejorar el *kernel* `RTM_IC_0_1` fue mínimo por su poco aporte al tiempo de ejecución global. Finalmente la mejora de los tres *kernels* representa una aceleración global de 3.258 veces. Este es el efecto combinado de los casos analizados anteriormente.

Tabla 17
Aceleración global al procesar un disparo con las versiones mejoradas de los kernels *en la estrategia 1. Se puede ver el efecto individual de la mejora en cada kernel y el efecto combinado de todas las mejoras.*

Kernel			**Tiempo [s]**			**Aceleración**		
`stencil_eval_gpu`	`apply_cpml_*_gpu`	`RTM_IC_0_1`	**Medido**	**Modelado**	**Error**	**Medido**	**Modelado**	**Error**
naive	*naive*	*naive*	1316.85	1766	25.42 %	1	1	0 %
mejorado	*naive*	*naive*	766.27	831.4	7.829 %	1.719	2.124	19.09 %
naive	*mejorado*	*naive*	957.32	1363	29.74 %	1.376	1.296	-6.143 %
naive	*naive*	*mejorado*	1314.22	1763	25.47 %	1.002	1.001	-0.06245 %
mejorado	*mejorado*	*mejorado*	404.15	425.7	5.061 %	3.258	4.148	21.45 %

El tiempo de ejecución de las estrategias 2 y 3 se presenta en las Tablas 18 y 19. Se observa que el impacto de mejorar el *kernel* `stencil_eval_gpu` es mayor al reportado en la estrategia 1 debido a que este *kernel* representa un mayor porcentaje sobre el tiempo de ejecución, como se observó en la Tabla 14. Los *kernels* relacionados con PML tienen un menor impacto en estas estrategias. En el caso de la estrategia 2, la reconstrucción del campo no requiere estos *kernels*, aunque si requiere el *kernel* `stencil_eval_gpu`; mientras que en la estrategia 3 no se requiere los *kernels* del PML en la propagación, ni en la reconstrucción del campo de la fuente. Finalmente el *kernel* `RTM_IC_0_1` sigue teniendo un impacto mínimo aunque ligeramente más relevante respecto a la estrategia 1.

Tabla 18
Aceleración global al procesar un disparo con las versiones mejoradas de los kernels *en la estrategia 2. Se puede ver el efecto individual de la mejora en cada kernel y el efecto combinado de todas las mejoras.*

Kernel			**Tiempo [s]**			**Aceleración**		
`stencil_eval_gpu`	`apply_cpml_*_gpu`	`RTM_IC_0_1`	**Medido**	**Modelado**	**Error**	**Medido**	**Modelado**	**Error**
naive	*naive*	*naive*	143.22	188.8	24.13 %	1	1	0 %
mejorado	*naive*	*naive*	75.46	78.98	4.454 %	1.898	2.39	20.59 %
naive	*mejorado*	*naive*	114.37	157.2	27.24 %	1.252	1.201	-4.269 %
naive	*naive*	*mejorado*	143.03	186.3	23.24 %	1.001	1.013	1.154 %
mejorado	*mejorado*	*mejorado*	45.87	44.96	-2.02 %	3.122	4.198	25.63 %

Tabla 19
Aceleración global al procesar un disparo con las versiones mejoradas de los kernels *en la estrategia 3. Se puede ver el efecto individual de la mejora en cada kernel y el efecto combinado de todas las mejoras.*

Kernel			**Tiempo [s]**			**Aceleración**		
`stencil_eval_gpu`	`apply_cpml_*_gpu`	`RTM_IC_0_1`	**Medido**	**Modelado**	**Error**	**Medido**	**Modelado**	**Error**
naive	*naive*	*naive*	125	164.8	24.15 %	1	1	0 %
mejorado	*naive*	*naive*	57.86	55.02	-5.168 %	2.16	2.996	27.88 %
naive	*mejorado*	*naive*	110.2	149	26.05 %	1.134	1.106	-2.559 %
naive	*naive*	*mejorado*	122.43	162.4	24.6 %	1.021	1.015	-0.5962 %
mejorado	*mejorado*	*mejorado*	40.77	36.8	-10.8 %	3.066	4.479	31.55 %

3.4.2. Estimación de la aceleración con los modelos.. En esta sección, se estimó el tiempo de ejecución del algoritmo para cada una de las estrategias con base en la información del análisis preliminar y a los modelos de predicción del tiempo de ejecución desarrollados para cada uno de los *kernels*. Así mismo se pudo determinar la aceleración al ejecutar las diferentes estrategias usando las versiones *naive* y

mejoradas de los *kernels*.

El tiempo de ejecución se estimó asumiendo que éste es la suma de los tiempos de ejecución de todos los *kernels* durante el desarrollo de cada estrategia. Esta afirmación es coherente teniendo en cuenta que:

- No se emplearon métodos para ejecutar varios *kernels* simultáneamente en la GPU debido a la dependencia de datos que existe en el proceso.
- La transferencia de datos es mínima durante todo el proceso debido a que para procesar el dato se restringió el algoritmo al uso de la memoria disponible en la GPU.
- Los procesos previos y posteriores al procesamiento del disparo no son significativamente costosos.

Basado en estas suposiciones, el tiempo de ejecución del algoritmo es:

$$t_{total} = \sum_{\forall kernels} [n_{kernel} * t_{kernel}] \tag{59}$$

donde n_{kernel} representa el número de veces que se ejecutó cada *kernel* y t_{kernel} es el tiempo de ejecución de un *kernel* específico. n_{kernel} se tomó directamente de la Tabla 14 aunque puede ser calculado para cada estrategia según el número de pasos de tiempo y otros parámetros de ejecución. Por otro lado t_{kernel} fue estimado mediante los modelos propuestos en este capítulo. Los tiempos estimados se presentan en las Tablas 17, 18 y 19 en la columna etiquetada como “Modelado”.

Se observa que el modelo tiene una tendencia a estimar un tiempo superior al medido, pero este error no supera el 30 % en las pruebas realizadas. Experimentalmente hay muchos factores que pueden afectar

el tiempo de ejecución que fueron minimizados pero no eliminados completamente como por ejemplo el entorno competitivo por los recursos generado por el sistema operativo, las optimizaciones al código por parte del compilador y el paralelismo o las estrategias de *scheduling* tanto en software como en hardware de los procesos, *threads*, etc. Pese a ello el resultado obtenido es consistente y permitió que el análisis detallado de cada *kernel* posibilitara la extrapolación de estos resultados hacia la aplicación global.

4. Conclusiones y discusión

4.1. Conclusiones

Se implementaron tres estrategias de manejo de memoria de la GPU para la reconstrucción del campo de la fuente y se establecieron las restricciones de memoria de cada estrategia.

La estrategia de puntos de control almacena el campo de la fuente en ciertos puntos a lo largo de la línea de tiempo, por lo que la información no almacenada del campo es recalculada desde el punto más cercano. A mayor cantidad de puntos de control menos información debe ser calculada nuevamente pero mayor será el consumo de memoria. En esta estrategia, la mejor configuración es la que emplea la memoria de la GPU para almacenar el mayor número de puntos de control porque disminuye el recómputo del campo y así mismo se reduce el tiempo de ejecución. La estrategia es una alternativa flexible en la implementación ya que se adapta a la memoria disponible en el sistema pero volver a calcular el campo incrementa significativamente su costo computacional.

La estrategia de retropropagación del campo de la fuente reduce considerablemente el tiempo de cómputo comparado con la estrategia de puntos de control (9.36 veces según datos de la Figura 29 y la Tabla 13 para el caso particular estudiado en el capítulo 2). Para ello el campo es recalculado hacia atrás a partir de un único punto de control ubicado al final de la línea de tiempo. La reconstrucción hacia atrás requiere recuperar la energía disipada por la frontera absorbente, por lo que es necesario almacenar varias capas del campo en los límites con la frontera CPML. Entonces la memoria requerida por esta estrategia debe distribuirse entre el punto de control en el último paso de tiempo y la frontera que debe ser almacenada. Por ello, esta estrategia no tiene ningún parámetro que permita ajustar la memoria empleada. Para el caso

de estudio del capítulo 2 la memoria requerida se acercó a los límites del dispositivo (2GB).

Finalmente la estrategia de fronteras aleatorias no requiere almacenar la frontera del campo, por lo que requiere la menor cantidad de memoria entre todas las estrategias (12.1 veces según datos de la Figura 29 y la Tabla 13 para el caso de estudio en el capítulo 2). Esto gracias a que la frontera aleatoria no disipa energía, solo la distorsiona. Además del ahorro de memoria, el tiempo de ejecución se reduce respecto a la estrategia de retropropagación del campo (1.35 veces según datos de la Figura 29 y la Tabla 13 para el caso de estudio en el capítulo 2) debido que no requiere emplear el algoritmo de fronteras absorbentes para el campo de la fuente.

El error en los resultados de la estrategia de retropropagación del campo de la fuente es por redondeo debido a la representación numérica de los datos. Éste mejora significativamente al emplear formato de punto flotante de precisión doble y mejora levemente al reducir el orden de aproximación de las diferencias finitas.

La retropropagación del campo de la fuente implica recalcular el campo aplicando de forma inversa las operaciones realizadas durante la propagación. Aunque desde la perspectiva algebraica el proceso es completamente reversible, el error debido a la representación numérica impide una reconstrucción perfecta. Este fenómeno se evidencia en el campo retropropagado como ruido y por ende en la condición de imagen para cada disparo y en el resultado final, al superponer los resultados de cada disparo.

Tomando como referencia el resultado de la estrategia de puntos de control se pudo aislar el error numérico. Otra alternativa de referencia era una solución analítica del problema, pero esta comparación añade

fuentes de error adicionales como el error propio del método numérico empleado. El formato de representación numérica de punto flotante de precisión doble mejoró el error calculado con la norma L2 normalizada en más de 20 órdenes de magnitud, como se ve en las Figuras 12 y 13, lo cual evidencia la alta sensibilidad de este parámetro para esta estrategia y de paso para la estrategia de fronteras aleatorias.

El orden de aproximación del método de diferencias finitas también influyó en el error, aunque en menor proporción. La tendencia general fue la de encontrar errores de menor magnitud con órdenes de aproximación menores. La principal razón de este fenómeno es que la aproximación de diferencias finitas de órdenes superiores implica un mayor número de operaciones matemáticas, por lo cual hay más ocasiones de redondeo.

El error de la estrategia de fronteras aleatorias no fue perceptible en la imagen final

Los resultados obtenidos en la Figura 19 muestran que para el caso de estudio se pudo aplicar exitosamente la estrategia de fronteras aleatorias. Aunque la norma L2 normalizada muestra un error de $3.970529e-03$, el resultado final es aceptable. La imagen migrada es la suma de los resultados de procesar cada uno de los disparos, los cuales a su vez son el resultado de sumar el producto entre el campo de la fuente y los receptores en todos los tiempos, por lo que el efecto del ruido aleatorio se reduce debido a la falta de coherencia.

Ciertas configuraciones de los parámetros `ks_rand` y `rand_mode` generan esquemas inestables que impiden la aplicación de la estrategia de fronteras aleatorias. Variar en el tiempo la velocidad de la frontera aleatoria empleando diferencias finitas no fue posible de forma segura.

El parámetro `ks_rand` se implementó basado en el artículo Fletcher and Robertsson (2011), el cual permite cambiar en el tiempo el valor de las velocidades al interior de la frontera aleatoria. `ks_rand` establece el periodo con el que estos valores son cambiados. Para evitar almacenar los valores de la velocidad cada vez que son cambiados, se implementó un generador lineal congruencial de números pseudoaleatorios para cada punto dentro de la frontera con parámetros diferentes. Cada una de las secuencias de valores aleatorios puede ser revertida sin error durante el proceso de retropropagación. En los experimentos realizados, se encontró que valores pequeños de `ks_rand` generan inestabilidad como se ve en la Tabla 11. Aunque en el artículo Fletcher and Robertsson (2011) no se reportó un fenómeno similar se presume que esto se debe a que el método numérico empleado allí fue *recursive integral time extrapolation*.

El parámetro `rand_mode` acota el rango de velocidades $[V_{min}V_{max}]$ para la generación de velocidades aleatorias según se especificó en la Tabla 2. Se encontró que este parámetro puede acelerar la rapidez con la que la energía crece en el campo para los esquemas inestables generados por `ks_rand`. Entre más grande sea el rango, más rápido se evidencia la inestabilidad del esquema.

Las configuraciones estables de estos parámetros se encuentran haciendo `ks_rand` $>$ `itmax`, es decir, sin variar la frontera aleatoria en todo el modelado. En estos casos el parámetro `rand_mode` no genera inestabilidad.

La mejor configuración encontrada para el parámetro `rand_mode` es la que emplea el rango completo de velocidades desde cero hasta el valor crítico de estabilidad de las diferencias finitas.

Para evaluar la efectividad de este parámetro en la frontera aleatoria se observó en el campo de la

fuente la energía coherente devuelta al interior del modelo, la cual se manifiesta en frentes de onda bien definidos. Existen dos frentes de onda que se pueden generar: (1) el que se forma por el cambio de impedancia acústica entre el modelo y la frontera aleatoria; y (2) el formado por la energía que entra y sale de la frontera aleatoria al reflejarse en el borde del espacio modelado. Para `rand_mode` $= 3$ se tiene un rango de valores aleatorios de velocidad simétrico respecto al valor de velocidad de referencia del modelo (V_{mod}) y esto permitió eliminar el frente (1) como se observó en la Figura 21 en el campo observado a los 2 segundos. Por otro lado la observación del campo a los 3 segundos en la Figura 22 permite ver que `rand_mode` $= 1$ distorsiona el frente de onda (2) mejor que las demás alternativas.

Debido a que el frente de onda (1) es de menor amplitud que el (2), se concluyó que `rand_mode` $= 1$ tiene el mejor desempeño combinado distorsionando los dos frentes de onda. Este modo emplea todo el rango de velocidades disponibles desde cero hasta el valor máximo que la estabilidad del esquema de diferencias finitas soporta. Esta alternativa puede generar velocidades por debajo del V_{min} definido en la Ecuación 22 que impide cumplir con el mínimo muestreo espacial por longitud de onda. Este mismo esquema fue empleado por Clapp (2009) y Fletcher and Robertsson (2011).

El incremento de la longitud de la frontera aleatoria (`L`) favorece el desempeño de la estrategia de frontera aleatoria.

El efecto de la frontera aleatoria permite a la estrategia 3 distorsionar el frente de onda que llega al borde del modelo. Un mayor grosor de la frontera (`L`) permite mejorar la acción de la misma en dos formas: (1) retrasar la energía que logra atravesar la frontera y que se refleja en el borde externo de la frontera; y (2) retener temporalmente parte de la energía al interior de la frontera que se convierte en una zona de

tránsito lento.

Se desarrolló un modelo de predicción del tiempo de ejecución de los *kernels* más relevantes de la implementación RTM 3D en función de: el tamaño del modelo, el orden de aproximación de las diferencias finitas y el tamaño de la frontera absorbente con un error absoluto menor al 30 %**.**

Los modelos de predicción del tiempo de ejecución se desarrollaron tanto para las versiones *naive* como para las versiones mejoradas de los *kernels* más relevantes de la implementación RTM. En todos los casos se pudo mantener la misma expresión matemática recalculando únicamente los parámetros del mismo. Estas expresiones fueron lineales respecto al tamaño del espacio que se procesa. El efecto del orden de aproximación de las diferencias finitas se pudo incluir dentro de la expresión matemática modificando los parámetros básicos del modelo lineal: la pendiente y el intercepto. Los modelos generados son adecuados para el rango de valores de entrada usados en la experimentación.

El efecto de los parámetros de hardware sobre el tiempo de ejecución no se ajustó a una expresión matemática, sino que su análisis se realizó estadísticamente sobre su tiempo de ejecución.

Los parámetros de hardware modifican el funcionamiento interno de la GPU. Para representar matemáticamente el impacto que tienen estos parámetros sobre el tiempo de ejecución se debe modelar el dispositivo desde el más bajo nivel comenzando por las transferencias de memoria, las instrucciones aritméticas y la dinámica de trabajo interno. Se consideró durante el desarrollo del presente trabajo que una caracterización descriptiva sería suficiente para acotar el tamaño del bloque de *threads* de la ejecución de cada *kernel* y así garantizar que el tiempo de ejecución será cercano a la mejor configuración posible.

Durante la ejecución del presente trabajo se exploró el desarrollo de un modelo con alto nivel de detalle comenzando por la caracterización de operaciones básicas, lo cual evidenció que este tipo de modelo requeriría un gran esfuerzo con varios factores que afectan la medición debido al desconocimiento de detalles no publicados sobre la arquitectura de la GPU, y que finalmente representaría simplemente un ajuste fino en la búsqueda del tiempo de ejecución óptimo.

Dentro de las acotaciones obtenidas sobre el tamaño del bloque de *threads*, se pueden observar restricciones coherentes con conceptos intuitivos del funcionamiento de la GPU. Por ejemplo, se favoreció el incremento de la dimensión rápida, lo cual permite mejor uso de recursos compartidos por los *threads* de un mismo *warp* facilitando accesos *coalesced*, evitando conflictos de banco, etc. En ese orden de ideas, también se destacó siempre el desempeño obtenido al asignar tamaños de bloque que corresponden a potencias de dos comparado con configuraciones similares.

En general hubo una tendencia a favorecer bloques grandes de *threads* donde se observó que las otras dos dimensiones espaciales no se deben dejar en su valor mínimo. Todo esto siempre bajo la restricción impuesta por el hardware, que impide que el tamaño del bloque sea superior a 1024 para esta arquitectura.

Los parámetros que más influyen en el tiempo de ejecución del algoritmo RTM 3D son: el tamaño del modelo, el orden de aproximación de las diferencias finitas, el grosor de la capa de las fronteras absorbentes.

Teniendo en cuenta que los modelos matemáticos desarrollados para estimar el tiempo de ejecución de la implementación en GPU del algoritmo lograron representar el comportamiento de los datos experi-

mentales, se pudo extraer la lista de los parámetros más relevantes de estos modelos. Así mismo se tuvo en cuenta que los *kernels* analizados representan más del 95 % del tiempo de ejecución del algoritmo.

4.2. Trabajo Futuro

La presente tesis doctoral, ha explorado diferentes aspectos de la implementación de algoritmos empleados en el procesamiento de datos sísmicos sobre plataformas de cómputo de memoria restringida como las GPU. En ese proceso se consolidan conceptos pero también se abren puertas para nuevos temas de investigación. En esta sección se dará un breve resumen de mi visión sobre el trabajo futuro.

Extensión a FWI. El algoritmo de *Full-Waveform Inversion* (FWI) es uno de los algoritmos que más interesan a la industria del gas y petróleo en la actualidad. Su ejecución tiene gran similitud con RTM aunque demanda más recursos computacionales. Por eso solo hasta hace unos pocos años se logró implementar. La adaptación de los resultados obtenidos durante el desarrollo de esta tesis es de gran interés con el fin de explotar el poder computacional de las GPU y emplear de forma óptima su memoria.

Extensión a otras ecuaciones de onda. La ecuación de onda empleada en esta tesis fue la ecuación de onda acústica con densidad constante. Pero existen formulaciones mucho más completas sobre el fenómeno de la propagación de la onda en diferentes medios. En cada caso los conceptos de RTM se conservan, aunque la condición de imagen, la condición de frontera y la aplicación del método numérico deben adaptarse. Desde el punto de vista computacional, su implementación requiere más memoria para almacenar más campos y más cómputo para procesar dichos campos. La metodología y las estrategias estudiadas en este documento son aplicables a estos casos pero se requiere trabajo adicional para solucionar los problemas específicos de cada uno de ellos.

Aplicación de la frontera aleatoria. A lo largo del documento se hizo un estudio de los parámetros implementados para la estrategia 3: Fronteras aleatorias. Este análisis observó el efecto que tienen los parámetros sobre la estabilidad del sistema y sobre la deformación del frente de onda; dado que el principal problema de la estrategia es la cantidad de ruido que produce en la imagen final. Sobre este aspecto

se debe realizar un estudio con mayor profundidad sobre los efectos de los parámetros sobre la imagen final y así mismo se deben determinar aspectos prácticos que garanticen un nivel de ruido aceptable para etapas posteriores a la migración sísmica.

Implementación de otras estrategias. Pueden existir estrategias adicionales o mejoras a las implementadas durante el desarrollo de la tesis. La estrategia 1 puede ser mejorada agregando los puntos de control óptimo. La estrategia 2 puede ser mejorada con conceptos desarrollados para el caso viscoacústico donde la retropropagación del campo en medios con pérdidas hace el sistema inestable. Ese mismo fenómeno se observa en la frontera absorbente CPML donde retropropagar la energía al interior del borde deriva en un sistema inestable. Con los conceptos del caso viscoacústico, no sería necesario para la estrategia 2 almacenar el borde del campo en cada paso de tiempo, sino que esta información es recuperada desde la propia frontera CPML, lo cual implica un ahorro muy significativo de memoria. La estrategia 3 podría ser mejorada o complementada con conceptos de otras estrategias para reducir la magnitud del ruido coherente.

Implementación de otros métodos numéricos. Durante la tesis solo se trabajó con diferencias finitas de orden variable, pero la combinación de las estrategias con otros métodos numéricos podría reducir los requerimientos de memoria o reducir el error y mejorar la calidad de los resultados.

Sobre la tecnología. Para la implementación del algoritmo no se buscó un dispositivo de última generación porque el objetivo de la tesis fue precisamente estudiar las restricciones de memoria y cómputo de la plataforma. La migración del presente trabajo para otras GPUs de mejores especificaciones es directa. La implementación de las estrategias sobre plataformas de múltiples GPU sobre un nodo o sobre un clúster de GPUs se exploró mediante una tesis de maestría que logró satisfactoriamente la implementación de una estrategia con dos formas de implementación en el clúster, pero hay muchos aspectos por trabajar.

Finalmente la implementación sobre plataformas alternativas de cómputo, como FPGA, requiere de mayor madurez de la tecnología y de las herramientas de desarrollo; en estos dispositivos puede ser aun más fuerte la restricción de memoria.

Implementación en la Industria. Las estrategias desarrolladas pueden ser empleadas en una implementación que se emplee a nivel industrial. Es necesario definir las especificaciones que tendría la implementación y el tiempo de desarrollo, pero seguramente el manejo óptimo de la memoria estudiado en esta tesis permitirá que esta implementación sea competitiva con otras disponibles.

Automatización de los modelos de predicción de tiempo de ejecución. La metodología propuesta y desarrollada en este trabajo, implica el desarrollo de modelos de predicción de tiempos de ejecución para las versiones originales y mejoradas de los *kernels*. Automatizar este proceso es posible y reduciría notablemente el tiempo de desarrollo. Esto implica analizar muchos más casos de uso de la metodología para la implementación automática.

Aplicación sobre otros datos. El procesamiento de otros conjuntos de datos, incluyendo datos reales, mediante las estrategias es muy interesante porque permitiría ver el desempeño de cada estrategia al procesar el dato y además se podría estudiar el impacto del error introducido por cada una de ellas en las imágenes finales.

4.3. Tesis dirigidas

Como apoyo al trabajo desarrollado en la presente investigación, se estudiaron temas relacionados con la tesis doctoral mediante tesis de maestría y pregrado. A continuación presento un pequeño resumen de estos trabajos:

Tesis de Maestría en Ingeniería Electrónica.

Título: Analytical model to estimate the execution time of a 3D acoustic wave equation implementation using FDTD in a GPU.

Autor: Dorfell Leonardo Parra Prada.

Resumen: Se desarrolló de un modelo analítico de predicción de tiempo de ejecución de la solución de la ecuación de onda en GPU y CPU. Se tomó el modelo *Memory Warp Parallelism-Compute Warp Parallelism* (MWP-CWP) Hong and Kim (2009); Sim et al. (2012) como referencia y se propuso una metodología para adaptar este modelo a un problema particular. El desarrollo de esta tesis permitió explorar modelos analíticos de predicción de tiempo de ejecución propuestos de tipo general y de aplicaciones específicas. Esta tesis a su vez se apoyó en las dos siguientes tesis de pregrado.

Tesis de Pregrado en Ingeniería Electrónica.

Título: Extracción de parámetros para el modelado del desempeño de una implementación GPU.

Autor: Andersson Nicolás Sánchez Gil.

Resumen: Esta tesis permitió determinar el valor de algunos parámetros de hardware de la GPU requeridos por el modelo *Memory Warp Parallelism-Compute Warp Parallelism* (MWP-CWP) Hong and Kim (2009); Sim et al. (2012). Durante el desarrollo de esta tesis se exploró el funcionamiento interno de la GPU mediante *benchmarks* que caracterizaron el tiempo de ejecución de operaciones básicas como una operación aritmética de punto flotante o el acceso a una posición de memoria global. La complejidad del comportamiento del tiempo de ejecución de operaciones tan sencillas, permitió ver cuán complejo puede ser modelar desde esta perspectiva analítica la ejecución de un *kernel* debido a políticas de funcionamiento interno de la GPU que no están documentadas completamente en la literatura.

Tesis de Pregrado en Ingeniería Electrónica.

Título: Predicción del tiempo de ejecución de una implementación *In-plane* de la ecuación de onda acústica 3D en una GPU usando un modelo analítico.

Autor: Xiomara Ribero Figueroa.

Resumen: En este trabajo se estudió el modelo propuesto por Tang et al. (2013) para la predicción del tiempo de ejecución de un *stencil* en una GPU y se intentó adaptar el modelo para predecir el tiempo de ejecución de la solución de la ecuación de onda. El artículo propuso una implementación con una nueva forma de cargar los datos desde la memoria global hacia la memoria compartida llamada *In-plane*; mientras que el modelo de predicción del tiempo de ejecución se basaba en los recursos críticos de la GPU que se agotaban durante a ejecución del algoritmo (e.g. registros, memoria compartida, *warps* por SM).

Tesis de Pregrado en Ingeniería Electrónica.

Título: Comparación de las herramientas de programación en paralelo OpenCL y CUDA para la implementación de la propagación de la ecuación de onda acústica 3D con densidad constante basada en stencil.

Autor: Johan Sebastian Suarez Largo.

Resumen: En este trabajo exploró las principales herramientas para el desarrollo de software en GPU: OpenCL y CUDA. Mediante la implementación de versiones básicas para la solución de la ecuación de onda acústica 3D se comparó el desempeño y la flexibilidad de estas herramientas. Finalmente los resultados permitieron definir CUDA como el lenguaje de programación que se empleó en los desarrollos de la tesis doctoral.

Tesis de Pregrado en Ingeniería Electrónica.

Título: Implementación del algoritmo de migración reversa en tiempo(RTM) 3D en CPU usando OpenMP.

Autor: Jesús Alonso Ortiz Lanzziano.

Resumen: Este trabajo permitió explorar el desempeño del algoritmo RTM 3D en una CPU empleando la librearía OpenMP que permite realizar ejecuciones *multi-thread*.

Tesis de Maestría en Ingeniería Electrónica.

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

Autor: Ivan Felipe Obregón Carreño.

Resumen: Mediante este trabajo se exploraron estrategias para ejecutar el algoritmo RTM en un entorno multi-GPU. Como primera estrategia distribuye los disparos que se van a procesar entre las GPUs disponibles del sistema para que cada una de ellas lo procese independientemente. Las estrategias estudiadas en esta tesis doctoral podrían ser empleadas para procesar cada uno de los disparos en cada GPU. La segunda estrategia, llamada descomposición de dominio, segmenta el modelo, asignando a cada GPU un espacio que debe actualizar en cada paso de tiempo. En esta estrategia participan las GPU disponibles en el procesamiento de cada uno de los disparos, por lo que la utilización de las estrategias analizadas en este trabajo de investigación deben ser adaptadas.

Referencias Bibliográficas

(2019). *DGX-2/2H System User Guide*. NVIDIA.

Alkhalifah, T. (2000). An acoustic wave equation for anisotropic media. *Geophysics*, 65(4):1239–1250.

Anderson, J. E., Tan, L., and Wang, D. (2012). Time-reversal checkpointing methods for RTM and FWI. *Geophysics*, 77(4).

Baysal, E., Kosloff, D. D., and Sherwood, J. W. C. (1983). Reverse time migration. *Geophysics*, 48(11):1514–1524.

Berenger, J.-P. (1994). A perfectly matched layer for the absorption of electromagnetic waves. *J. Comput. Phys.*, 114(2):185–200.

Berenger, J.-p. (1996). Perfectly Matched Layer for the FDTD Solution of Wave-Structure Interaction Problems. *IEEE Transactions on Antennas and Propagation*, 44(1):110–117.

Bleistein, N. (1987). On the imaging of reflectors in the earth. *GEOPHYSICS*, 52(7):931–942.

Bo, F. and Huazhong, W. (2012). Reverse time migration with source wavefield reconstruction strategy. *Journal of Geophysics and Engineering*, 9(1):69–74.

Chang, W.-F. and McMechan, G. A. (1987). Elastic reverse-time migration. *Geophysics*, 52(10):1365–1375.

Chew, W. C. and Liu, Q. H. (1996). Perfectly Matched Layers for Elastodynamics: A New Absorbing Boundary Condition. *Journal of Computational Acoustics*, 4(4):341–359.

Claerbout, J. F. (1971). Toward a unified theory of reflector mapping. *Geophysics*, 36(June):467–481.

Clapp, R. G. (2009). Reverse time migration with random boundaries SEG Houston 2009 International Exposition and Annual Meeting SEG Houston 2009 International Exposition and Annual Meeting. pages 2809–2813.

Clapp, R. G., Haohuan Fu, and Lind, O. (2010). Selecting the right hardware for reverse time migration. *The Leading Edge*, 29(1):48–58.

Dussaud, E., Total, E., Symes, W. W., Williamson, P., Lemaistre, L., Singer, P., Denel, B., Cherrett, A., and Sa, T. (2008). Computational strategies for reverse-time migration. In *SEG Las Vegas 2008 Annual Meeting*, pages 2267–2271.

Fletcher, R. P. and Robertsson, J. O. (2011). Time-varying boundary conditions in simulation of seismic wave propagation. *Geophysics*, 76(1):A1–A6.

Fornberg, B. (1987). The pseudospectral method: Comparisons with finite differences for elastic wave ecuation. *Geophysics*, 52(4):483–501.

Gazdag, J. (1978). Wave equation migration with the phase-shift method. *Geophysics*, 13(7):1342–1351.

Gazdag, J. and Sguazzero, P. (1984). Migration of seismic data by phase shift plus interpolation. *Geophysics*, 49(2):124–131.

Griewank, A. (1992). Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*.

Guitton, A., Valenciano, A., Bevc, D., and Claerbout, J. (2006). Robust illumination compensation for shot-profile migration.

Hong, S. and Kim, H. (2009). An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. *ACM SIGARCH Computer Architecture News*, 37(3):152–163.

Liu, H.-W., Li, B., Liu, H., Tong, X.-L., and Liu, Q. (2010). The Algorithm of High Order Finite Difference Pre-Stack Reverse Time Migration and GPU Implementation. *Chinese Journal of Geophysics*, 53(4):600–610.

NVIDIA Corporation (2013). CUDA C programming guide.

NVIDIA Corporation (2016). CUDA C best practices guide.

Pasalic, D. and McGarry, R. (2010). Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations. *2010 SEG Annual Meeting*, (2).

Sim, J., Dasgupta, A., Kim, H., and Vuduc, R. (2012). A performance analysis framework for identifying potential benefits in gpgpu applications. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 11–22, New York, NY, USA. ACM.

Stoffa, P. L., Fokkema, J. T., de Luna Freire, R. M., and Kessinger, W. P. (1990). Split-step Fourier migration. *Geophysics*, 55(4):410–421.

Symes, W. W. (2007). Reverse time migration with optimal checkpointing. *Geophysics*, 72(5):SM213–SM221.

Tang, W. T., Tan, W. J., Krishnamoorthy, R., Wong, Y. W., Kuo, S. H., Goh, R. S. M., Turner, S. J., and Wong, W. F. (2013). Optimizing and auto-tuning iterative stencil loops for gpus with the in-plane method. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 452–462.

Vivas, F. and Pestana, R. (2007). Imaging condition to true-amplitude shot-profile migration. *Seg Technical Program Expanded Abstracts*, 26.

Vivas, F. A., Pestana, R. C., and Ursin, B. (2009). A new stabilized least-squares imaging condition. *Journal of Geophysics and Engineering*, 6(3):264–268.

Whitmore, N. (1983). Iterative depth migration by backward time propagation. *SEG Technical Program Expanded Abstracts*, pages 382–385.

Yoon, K., Shin, C., Suh, S., Lines, L. R., and Hong, S. (2003). 3D reverse-time migration using the acoustic wave equation : An experience with the SEG / EAGE data set. *The Leading Edge*, (Figure 12):38–41.

Zhang, W. and Shen, Y. (2010). Unsplit complex frequency-shifted PML implementation using auxiliary differential equations for seismic wave modeling. *Geophysics*, 75(4):T141–T154.

Zhou, H., Zhang, G., and Bloor, R. (2006). An anisotropic acoustic wave equation for modeling and migration in 2d tti media. In *SEG Technical Program Expanded Abstracts 2006*, pages 194–198. Society of Exploration Geophysicists.

Zhu, J. and Lines, L. R. (1998). Comparison of Kirchhoff and reverse-time migration methods with applications to prestack depth imaging of complex structures. *Geophysics*, 63(4):1166–1176.

Apéndices

Apéndice A. Ecuaciones CPML

En este apéndice se presentará una interpretación propia de la deducción de las ecuaciones empleadas en la implementación de *Convolutional Perfectly Matched Layers* (CPML). Esta interpretación se basa en el trabajo de Pasalic and McGarry (2010).

CPML modifica la ecuación de onda para introducir atenuación en los bordes del modelo y así evitar reflexiones artificiales en los mismos. Para ello, los operadores diferenciales de la ecuación de onda se someten a un factor de escala complejo en el dominio de la frecuencia. Este factor:

$$s_i = 1 + \frac{\sigma_i}{\alpha_i + j\omega}, \tag{60}$$

$$\frac{1}{s_i} = 1 - \frac{\sigma_i}{\sigma_i + \alpha_i + j\omega}, \tag{61}$$

se aplica a la ecuación de onda acústica (16) en cada una de las direcciones $i \in \{x, y, z\}$. σ_i y α_i son parámetros que ajustan el comportamiento de la frontera absorbente. El operador diferencial se sustituye por una versión escalada así:

$$\frac{\partial}{\partial i} \rightarrow \mathscr{F}^{-1}\left\{\frac{1}{s_i}\right\} * \frac{\partial}{\partial i}, \tag{62}$$

donde $\mathscr{F}^{-1}$ denota la transformada inversa de Fourier. Expandiendo este factor se obtiene

$$\mathscr{F}^{-1}\left\{\frac{1}{s_i}\right\} = \mathscr{F}^{-1}\left\{1 - \frac{\sigma_i}{\sigma_i + \alpha_i + j\omega}\right\} = \delta(t) - \sigma_i e^{-(\sigma_i+\alpha_i)t}\mu(t), \tag{63}$$

donde $\delta(t)$ representa la función delta de Dirac y $\mu(t)$ la función escalón unitario. Por lo tanto la sustitución de los operadores diferenciales queda

$$\frac{\partial}{\partial i} \rightarrow \frac{\partial}{\partial i} + \psi_i \qquad \text{donde} \qquad \psi_i = -\sigma_i e^{-(\sigma_i+\alpha_i)t}\mu(t) * \frac{\partial}{\partial i}. \tag{64}$$

Aplicando la definición de convolución sobre el término ψ_i se tiene,

$$\psi_i = -\int_{-\infty}^{\infty} \sigma_i e^{-(\sigma_i+\alpha_i)(t-\tau)}\mu(t-\tau)\left(\frac{\partial}{\partial i}\right)^{\tau} \tag{65}$$

donde $\left(\frac{\partial}{\partial i}\right)^{\tau}$ representa el operador diferencial evaluado en el punto $t = \tau$. Algunos elementos que no son función del tiempo pueden ser extraídos de la integral.

$$\psi_i = -\sigma_i e^{-(\sigma_i+\alpha_i)t}\int_{-\infty}^{\infty} e^{(\sigma_i+\alpha_i)\tau}\mu(t-\tau)\left(\frac{\partial}{\partial i}\right)^{\tau} \tag{66}$$

En términos prácticos, el operador diferencial $\left(\frac{\partial}{\partial i}\right)^{\tau}$ solo podrá ser evaluado en puntos discretos de tiempo. Para este caso, haciendo un muestreo espacial uniforme ($\tau = \{\Delta t, 2\Delta t, 2\Delta t, \ldots, n\Delta t\}$), se supondrá que este operador tendrá un valor constante en intervalos de Δt como lo muestra la figura 58. De la misma forma, teniendo en cuenta el interés por calcular la función ψ_i para los mismos tiempos discretos y haciendo una interpretación gráfica de la convolución como en la figura 58 se puede calcular la integral por intervalos

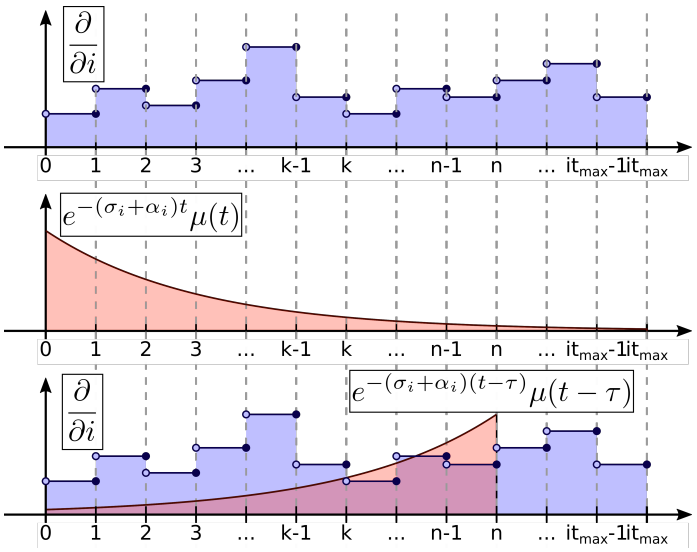


Figura 58. Representación gráfica de las funciones involucradas en la convolución de la ecuación. (a) Operador diferencial. (b) Función exponencial. (c) Cálculo de la convolución

Δt donde el valor del operador $\left(\frac{\partial}{\partial i}\right)^{\tau}$ no depende del tiempo.

$$\psi_i^n = -\sigma_i e^{-(\sigma_i+\alpha_i)n\Delta t} \sum_{k=1}^{n} \int_{(k-1)\Delta t}^{(k)\Delta t} e^{(\sigma_i+\alpha_i)\tau} \left(\frac{\partial}{\partial i}\right)^k \tag{67}$$

Resolviendo la integral

$$\psi_i^n = -\sigma_i e^{-(\sigma_i+\alpha_i)n\Delta t} \sum_{k=1}^{n} \frac{1}{\sigma_i+\alpha_i} e^{(\sigma_i+\alpha_i)\tau}\Big|_{(k-1)\Delta t}^{(k)\Delta t} \left(\frac{\partial}{\partial i}\right)^k \tag{68}$$

$$\psi_i^n = -\sigma_i e^{-(\sigma_i+\alpha_i)n\Delta t} \sum_{k=1}^{n} \frac{1}{\sigma_i+\alpha_i} \left[e^{(\sigma_i+\alpha_i)k\Delta t} - e^{(\sigma_i+\alpha_i)(k-1)\Delta t}\right] \left(\frac{\partial}{\partial i}\right)^k \tag{69}$$

Sea $a_i = e^{-(\sigma_i+\alpha_i)\Delta t}$,

$$\psi_i^n = -\sigma_i a_i^n \sum_{k=1}^{n} \frac{1}{\sigma_i+\alpha_i} \left[a_i^{-k} - a_i^{-k+1}\right] \left(\frac{\partial}{\partial i}\right)^k \tag{70}$$

$$\psi_i^n = \sigma_i a_i^n \frac{1}{\sigma_i + \alpha_i} (a_i - 1) \sum_{k=1}^{n} a_i^{-k} \left(\frac{\partial}{\partial i}\right)^k \tag{71}$$

Extrayendo el último término de la sumatoria,

$$\psi_i^n = \sigma_i a_i^n \frac{1}{\sigma_i + \alpha_i} (a_i - 1) \left[a_i^{-n} \left(\frac{\partial}{\partial i}\right)^n + \sum_{k=1}^{n-1} a_i^{-k} \left(\frac{\partial}{\partial i}\right)^k \right] \tag{72}$$

$$\psi_i^n = \sigma_i \frac{1}{\sigma_i + \alpha_i} (a_i - 1) \left(\frac{\partial}{\partial i}\right)^n + a_i \left[\sigma_i a_i^{n-1} \frac{1}{\sigma_i + \alpha_i} (a_i - 1) \sum_{k=1}^{n-1} a_i^{-k} \left(\frac{\partial}{\partial i}\right)^k \right] \tag{73}$$

Haciendo $b_i = \frac{\sigma_i}{\sigma_i + \alpha_i}(a_i - 1)$ y cómparando el término al interior del corchete con la ecuación 71, se observa que éste corresponde al mismo término ψ_i evaluado en $n-1$. Por lo tanto se tiene una relación recursiva para ψ_i^n:

$$\psi_i^n = b_i \left(\frac{\partial}{\partial i}\right)^n + a_i \psi_i^{n-1} \tag{74}$$

El operador de la segunda derivada se calcula aplicando el operador de la primera derivada recursivamente:

$$\frac{\partial}{\partial i}\left(\frac{\partial}{\partial i}\right) \rightarrow \frac{\partial}{\partial i}\left(\frac{\partial}{\partial i} + \psi_i\right) + \zeta_i \tag{75}$$

$$\frac{\partial^2}{\partial i^2} \rightarrow \frac{\partial^2}{\partial i^2} + \frac{\partial}{\partial i}\psi_i + \zeta_i \tag{76}$$

El operador ζ_i es análogo a ψ_i pero la expresión para actualizarlo recursivamente debe ser ajustado

para el argumento del segundo operador.

$$\zeta_i^n = a_i \zeta_i^{n-1} + b_i \left[\left(\frac{\partial^2}{\partial i^2} \right)^n + \left(\frac{\partial \psi_i}{\partial i} \right)^n \right] \tag{77}$$

Con las definiciones recursivas de las ecuaciones 74 y 77 se pueden actualizar en cada paso de tiempo los campos ψ_i y ζ_i. Estos campos a su vez requieren la actualización del campo de la fuente en cada paso de tiempo, por lo que se crea una relación que requiere actualizarlos alternadamente para avanzar en la extrapolación en el tiempo del campo de onda.

Apéndice B. Implementación RTM

La implementación de RTM se desarrolló en lenguaje C/CUDA con el objetivo de tener control a un nivel suficientemente bajo de las tareas que se ejecutan en el sistema de cómputo y además sacar provecho del *framework* dado por el fabricante de las GPU. Como principios generales se tuvo:

Modularidad: Se buscó que el código permitiera explorar optimizaciones en diferentes aspectos del algoritmo como por ejemplo las estrategias, el método numérico, el formato de representación de punto flotante, la condición de imagen, diferentes tipos de procesamiento final de imagen, etc. Por ello se estructuró el código para ofrecer flexibilidad mediante la segmentación del mismo en rutinas y librerías que permiten la reutilización de código y la adición de nuevas funciones.

Portabilidad: Para compilar el código no se requiere más que el entorno de desarrollo de CUDA, Seismic Un*x (SU) y las herramientas estándar de Linux para desarrollo. La ejecución requiere de un equipo con una GPU Nvidia con los drivers configurados y no se emplearon sentencias exclusivas de alguna arquitectura de GPU específica.

Marco Seismic Un*x: Se empleó el contexto de funciones que brinda SU para el desarrollo del código. Esto permite usar los formatos estándar de SU y las funciones de entrada y salida de datos. La sintaxis de la línea de comandos es cómoda para un usuario familiarizado con SU y eventualmente permitiría una fácil adaptación de una interfaz de usuario.

Plataformas hardware

La plataforma empleada para el desarrollo y las pruebas del algoritmo así como el modelo del tiempo de ejecución fue un equipo portátil ASUS modelo ROG GL752VW cuyas especificaciones se presentan en la Tabla 20.

Tabla 20
Especificaciones del equipo de pruebas.

Característica	**Valor**
Referencia CPU	Intel(R) Core(TM) i7-6700HQ
Cantidad de núcleos	4
Cantidad de subprocesos	8
Frecuencia del procesador	[2.6 - 3.5] GHz
Memoria RAM	16 GB DDR4 2133 MHz
Sistema operativo	Linux Debian 9.0 Stretch

Estructura general

La implementación inicialmente trabaja sobre los datos de entrada y los adecúa para garantizar que los procesos de modelado y migración se lleven a cabo adecuadamente. Posteriormente comienza el procesamiento de disparos con las opciones que el usuario ha ingresado. La figura 59 muestra el diagrama de flujo general de la implementación.

Declaración de variables. El programa inicia con la declaración de todas las variables requeridas durante la ejecución del algoritmo, antes de iniciar el procesamiento. Se destacan dos tipos de variables dentro del código:

Precisión de punto flotante. El código se puede compilar para trabajar con punto flotante de precisión simple (`float`) o precisión doble (`double`). Para ello se utiliza el macro `ftype` que es sustituido según se define en el archivo Makefile. Algunas variables no tienen la posibilidad de ser repre-

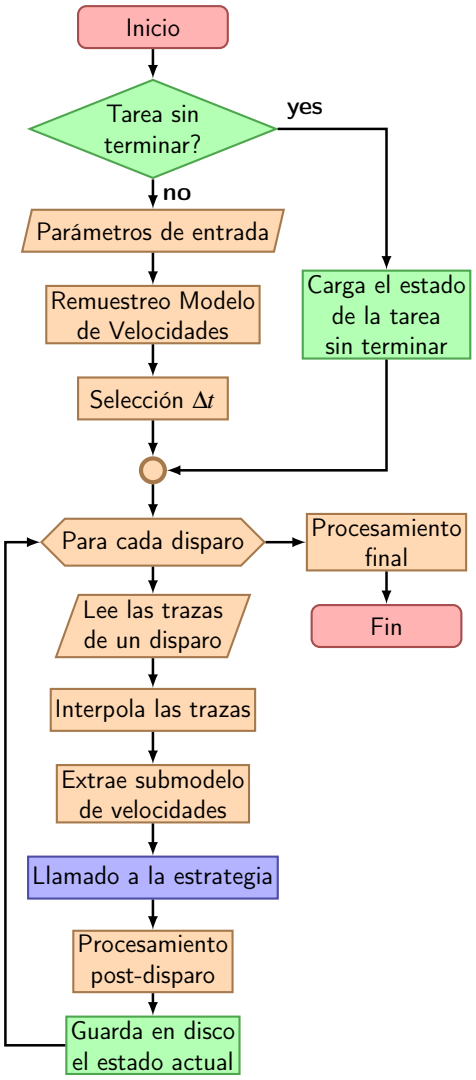


Figura 59. Diagrama de flujo de la implementación RTM

sentadas en precisión doble, porque se consideró que no lo requerían, por ejemplo, las relacionadas con el modelo de velocidades.

Estructuras de datos. Se crearon algunas estructuras de datos que facilitaron el acceso a la información. Se destacan la estructura `wavelet` que almacena la fuente y las trazas junto con sus respectivas posiciones en el modelo y la estructura `boundary` que almacena las fronteras en las estrategias que así lo requirieron.

Estado de ejecución. Con el fin de no perder resultados parciales durante la ejecución del proceso completo, cada vez que un disparo es procesado, se guarda un punto de restauración del proceso. En caso de falla al inicio de la ejecución se verifica si hubo una ejecución inconclusa, tras lo cual se restauran todas las variables y se continúa con el proceso. En la figura 59 se observan los bloques relacionados en color verde.

Lectura de datos de entrada. La lectura de los datos desde la línea de comandos se realiza mediante los macros que SU tiene dispuesto para ello. La lista completa de parámetros de entrada leídos se encuentra en la tabla 21

Opción en línea de comandos	**Variable C**	**Descripción**
`smovie=1/0`	flags	Crea un archivo binario del campo de la fuente por cada paso de tiempo en el proceso de modelado.
`sbackmovie=1/0`	flags	Crea un archivo binario del campo de la fuente por cada paso de tiempo en la retropropagación.
`rmovie=1/0`	flags	Crea un archivo binario del campo de los receptores por cada paso de tiempo en la retropropagación.
`resampfile=1/0`	flags	Crea un archivo formato SU con las trazas remuestreadas. Un archivo por cada disparo procesado.
`velresampfile=1/0`	flags	Crea un archivo binario con el modelo de velocidades remuestreado.
`revolutionfile=1/0`	flags	Crea un archivo binario en cada disparo con el resultado parcial del mapa de reflectividad.
`rshotfile=1/0`	flags	Crea un archivo binario con el resultado de procesar cada uno de los disparos con procesamiento post-disparo (si lo hay).

Tabla 21
Argumentos de entrada del programa

Opción en línea de comandos	**Variable C**	**Descripción**
`rshotfile_raw=1/0`	flags	Crea un archivo binario con el resultado de procesar cada uno de los disparos sin procesamiento post-disparo.
`rfile=1/0`	flags	Crea el archivo binario `r_file` con mapa de reflectividad con procesamiento post-migración.
`rfile_raw=1/0`	flags	Crea el archivo binario `r_file_raw` con mapa de reflectividad sin procesamiento post-migración.
`subvelfile=1/0`	flags	Crea un archivo binario con el submodelo de velocidades utilizado en cada uno de los disparos procesados.
`rdfile=1/0`	flags	Crea un archivo de texto con los valores de la función envolvente para la generación del modelo con fronteras aleatorias. (Válido solo para la estrategia 7)
`velrand1file=1/0`	flags	Crea un archivo binario con los modelos de velocidades empleados durante el proceso de modelado de la fuente. (Válido solo para la estrategia 7)
`velrand2file=1/0`	flags	Crea un archivo binario con los modelos de velocidades empleados durante el proceso de retropropagación de la fuente. (Válido solo para la estrategia 7)
`velextfile=1/0`	flags	Crea un archivo binario con el modelo de velocidades completo extendido.
`seffmovie=1/0`	flags	Crea un archivo binario del campo de la fuente por cada paso de tiempo en el proceso de modelado. Este archivo tiene el mismo tamaño que el resultado de procesar un disparo.
`seffbackmovie=1/0`	flags	Crea un archivo binario del campo de la fuente por cada paso de tiempo en la retropropagación. Este archivo tiene el mismo tamaño que el resultado de procesar un disparo.
`make_modeling=1/0`	flags	Habilita la propagación de la fuente. (Solo implementado en la estrategia 7)

Tabla 21
Argumentos de entrada del programa

Opción en línea de comandos	**Variable C**	**Descripción**
`make_retropropagation=1/0`	flags	Habilita la reconstrucción hacia atrás del campo de la fuente. (Solo implementado en la estrategia 7)
`make_backpropagation=1/0`	flags	Habilita la retropropagación del campo de los receptores. (Solo implementado en la estrategia 7)
`CUDA_dev=<int>`	CUDA_dev	Selecciona cual dispositivo CUDA se usará durante el procesamiento. El índice se puede consultar con el comando `nvidia-smi`
`prop_stencil_kernel=<int>`	prop_stencil_kernel	Selecciona cuál de los kernel para propagación se va a usar. Las opciones son: ▪ 0: No hace nada ▪ 1: ztencil_eval_gpu ▪ 2: stencil_eval_gpu_25_inplane ▪ 3: stencil_eval_gpu_opt ▪ 4: stencil_eval_gpu_opt_shm ▪ 5: stencil_eval_gpu (Versión *naive*) ▪ 6: stencil_eval_gpu_6 ▪ 7: stencil_eval_gpu_7 ▪ 8: stencil_eval_gpu_8 ▪ 9: stencil_eval_gpu_9 (Versión *mejorada*)
`pml_stencil_kernel=<int>`	pml_stencil_kernel	Selecciona cuál de los kernel para PML se va a usar. Las opciones son: ▪ 1: Kernel versión *naive*. ▪ 2: Kernel versión *mejorada*.
`psgraphfile=1/0`	flags	Crea un archivo para visualizar desde la superficie, el área de los submodelos de cada uno de los disparos. (Será eliminada esta opción)
`met=<int>`	num_method	Método numérico. Por ahora solo se usa diferencias finitas.

Tabla 21
Argumentos de entrada del programa

Opción en línea de comandos	**Variable C**	**Descripción**
`ord=<int>`	ord	Orden de aproximación para las diferencias finitas. Debe ser un número par.
`dx=<float>`	userdx	Resolución del modelo de velocidades en la dirección x [m].
`dy=<float>`	userdy	Resolución del modelo de velocidades en la dirección y [m].
`dz=<float>`	userdz	Resolución del modelo de velocidades en la dirección z [m].
`nx=<int>`	userFNx	Tamaño del modelo de velocidades en la dirección x [m].
`ny=<int>`	userFNy	Tamaño del modelo de velocidades en la dirección y [m].
`nz=<int>`	userFNz	Tamaño del modelo de velocidades en la dirección z [m].
`vfile=<ruta>`	vel_file	Ruta hacia el archivo binario del modelo de velocidades. Tamaño del archivo esperado es nx*ny*nz*4 bytes.
`vcte=<float>`	vcte	En caso de no incluir archivo de velocidades, se puede generar un modelo de velocidad constante `vcte` [m/s].
`lext=<float>`	lext	Extensión del modelo completo en la cara `Left`. Ver Sección 2.
`rext=<float>`	rext	Extensión del modelo completo en la cara `Right`. Ver Sección 2.
`bext=<float>`	bext	Extensión del modelo completo en la cara `Back`. Ver Sección 2.
`fext=<float>`	fext	Extensión del modelo completo en la cara `Front`. Ver Sección 2.
`text=<float>`	text	Extensión del modelo completo en la cara `Top`. Ver Sección 2.
`oext=<float>`	oext	Extensión del modelo completo en la cara `bOttom`. Ver Sección 2.
`fq=<float>`	fq	Frecuencia central de la ondícula Ricker.
`lpad=<float>`	lpad	Apertura para la migración de cada disparo en el lado `Left`. Ver Sección 2.

Tabla 21
Argumentos de entrada del programa

Opción en línea de comandos	**Variable C**	**Descripción**
`rpad=<float>`	rpad	Apertura para la migración de cada disparo en el lado `Right`. Ver Sección 2.
`bpad=<float>`	bpad	Apertura para la migración de cada disparo en el lado `Back`. Ver Sección 2.
`fpad=<float>`	fpad	Apertura para la migración de cada disparo en el lado `Front`. Ver Sección 2.
`tpad=<float>`	tpad	Apertura para la migración de cada disparo en el lado `Top`. Ver Sección 2.
`opad=<float>`	opad	Apertura para la migración de cada disparo en el lado `bOttom`. Ver Sección 2.
`strategy=<int>`	strategy	Selecciona la estrategia a utilizar: ▪ 0: 2D Almacena todo el campo. (Depreciada) ▪ 1: Almacena todo el campo en RAM de la CPU. Procesa en CPU. **(Depreciada)** ▪ 2: Almacena todo el campo en RAM de la CPU. Procesa en GPU. **(Depreciada)** ▪ 3: Almacena fronteras en RAM de la CPU. Reconstruye el campo de la fuente hacia atrás en GPU. **(Depreciada)** ▪ 4: Reconstruye el campo de la fuente hacia atrás en GPU sin guardar la frontera. **(Inestable) (Depreciada)** ▪ 5: Almacena todo el campo en RAM de la GPU. Procesa en GPU. **(Depreciada)** ▪ 6: Almacena las fronteras en GPU. Reconstruye el campo de la fuente hacia atrás en GPU **(Estrategia 2 del documento)**. ▪ 7: Fronteras aleatorias **(Estrategia 3 del documento)**. ▪ 8: Guarda puntos de control en GPU. Procesa en la GPU. **(Estrategia 1 del documento)**.

Tabla 21
Argumentos de entrada del programa

Opción en línea de comandos	**Variable C**	**Descripción**
`ks=<int>`	ks	Periodo para calcular la condición de imagen.
`ks_store=<int>`	ks_store	Periodo para almacenar los puntos de control de la estrategia 1.
`ks_rand=<int>`	ks_rand	Periodo para cambiar las fronteras aleatorias del modelo.(Válido solo para la estrategia 3)
`rdtype=<int>`	rdtype	Selecciona la función envolvente para la generación de números aleatorios. ▪ 0: Lineal ▪ 1: Exponencial ▪ 2: Cuadrática ▪ 3: Cero ▪ 4: Uno
`rand_mode=<int>`	rand_mode	Selecciona el rango de los valores de velocidad aleatoria. ▪ 0: Rango completo. ▪ 1: Mínimo cumpliendo Nyquist. ▪ 2: Mínimo cumpliendo cuatro veces Nyquist. ▪ 3: Rango simétrico.

Tabla 21
Argumentos de entrada del programa

Opción en línea de comandos	**Variable C**	**Descripción**
`imgcond=<int>`	imgcond	Selecciona la condición de imagen a utilizar. ▪ 0: Tradicional (Versión *naive*) ▪ 1: Normalizada con el campo de la fuente en cada disparo. ▪ 2: Normalizada con el campo de los receptores en cada disparo. ▪ 3: Normalizada con el campo de la fuente al final. ▪ 4: Normalizada con el campo de los receptores al final. ▪ 5: Suma de las condiciones de imagen 3 y 4. ▪ 7: Tradicional (Versión *mejorada*).
`imcondeps=<float>`	imcondeps	Parámetro para algunas condiciones de imagen.
`art_rem_method=<int>`	art_rem_method	Selecciona el método de procesamiento post condición de imagen. ▪ 0: Ninguno ▪ 1: Filtro laplaciano en cada uno de los disparos. ▪ 2: Filtro laplaciano al final de todos los disparos.
`pplo=<float>`	pplo	Puntos por longitud de onda (Muestreo espacial).
`abc=<1/0>x6`	abc	Selecciona las caras donde habrá condición de frontera.
`Lpml=<int>`	L	Capas de la condición de frontera.
`data=<ruta>`	data_file	Archivo de las trazas de entrada.
`dt=<float>`	dt	Paso de tiempo a utilizar en el procesamiento.
`nshots=<int>`	nshots	Cantidad de disparos a procesar.
`incshot=<int>`	incshot	Incremento en el índice de disparos a procesar.

Tabla 21
Argumentos de entrada del programa

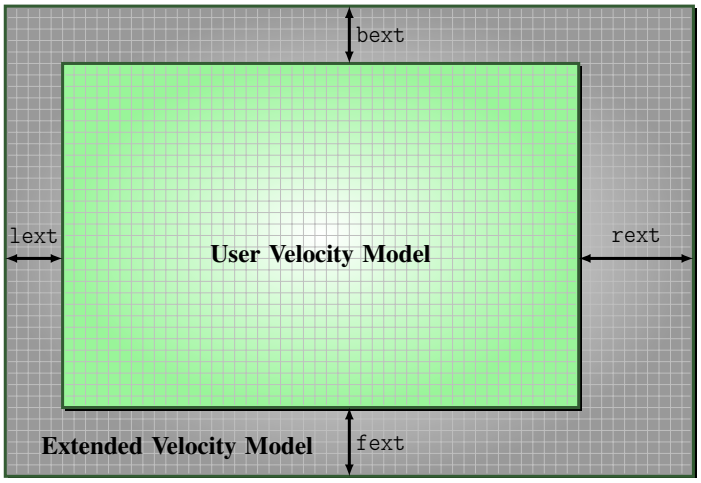


Figura 60. Vista superior del proceso de extensión del modelo de velocidades.

Opción en línea de comandos	**Variable C**	**Descripción**
`ishot=<int>`	ishot	Primer disparo a procesar.

Tabla 21
Argumentos de entrada del programa

Extensión del modelo de velocidades. La implementación del algoritmo permite extender el modelo de velocidades dado por el usuario. Los parámetros `lext`, `rext`, `bext`, `fext`, `text` y `oext` controlan la distancia extra en cada una de las direcciones. La velocidad en estos puntos es dada por el valor conocido más cercano. La figura 60 muestra la vista superior de este proceso. El tamaño del modelo está dado por:

$$\texttt{userFNx\_ext} = \texttt{userFNx} + \left\lfloor \frac{\texttt{lext}}{\texttt{userdx}} \right\rfloor + \left\lfloor \frac{\texttt{rext}}{\texttt{userdx}} \right\rfloor \tag{78}$$

$$\texttt{userFNy\_ext} = \texttt{userFNy} + \left\lfloor \frac{\texttt{bext}}{\texttt{userdy}} \right\rfloor + \left\lfloor \frac{\texttt{fext}}{\texttt{userdy}} \right\rfloor \tag{79}$$

$$\texttt{userFNz\_ext} = \texttt{userFNz} + \left\lfloor \frac{\texttt{text}}{\texttt{userdz}} \right\rfloor + \left\lfloor \frac{\texttt{oext}}{\texttt{userdz}} \right\rfloor \tag{80}$$

Ajuste de los puntos por longitud de onda. Para garantizar bajos niveles de dispersión numérica, el usuario puede controlar el muestreo espacial del campo de onda $(\Delta x, \Delta y, \Delta z)$. Esto se hace mediante el parámetro puntos por longitud de onda (`pplo`). El algoritmo debe remuestrear el modelo de velocidades teniendo en cuenta la frecuencia de la ondícula f_q así:

$$\Delta s = \frac{V_{min}}{\texttt{pplo} * f_q} \tag{81}$$

donde Δs es el máximo valor que puede tomar Δx, Δy y Δz. Adicionalmente el algoritmo toma en cuenta la localización de las trazas y las fuentes para garantizar que siempre estén siempre sobre un punto de grilla. Esto lo hace forzando que factor de sobremuestreo `factorx`, `factory` y `factorz` del modelo sean enteros:

$$\texttt{factorx} = \left\lceil \frac{\texttt{userdx}}{\Delta s} \right\rceil \qquad \texttt{factory} = \left\lceil \frac{\texttt{userdy}}{\Delta s} \right\rceil \qquad \texttt{factorz} = \left\lceil \frac{\texttt{userdz}}{\Delta s} \right\rceil \tag{82}$$

$$\texttt{dx} = \frac{\texttt{userdx}}{\texttt{factorx}} \qquad \texttt{dy} = \frac{\texttt{userdy}}{\texttt{factory}} \qquad \texttt{dz} = \frac{\texttt{userdz}}{\texttt{factorz}}. \tag{83}$$

El nuevo tamaño del modelo está dado por:

$$\begin{aligned} \texttt{FNx} &= (\texttt{userFNx\_ext} - 1)\texttt{factorx} + 1 \\ \texttt{FNy} &= (\texttt{userFNy\_ext} - 1)\texttt{factory} + 1 \\ \texttt{FNz} &= (\texttt{userFNz\_ext} - 1)\texttt{factorz} + 1. \end{aligned} \tag{84}$$

Selección de Δt. La selección del paso de tiempo Δt tiene como mayor prioridad mantener la estabilidad del esquema de propagación (Ecuación 21). De igual forma, se tuvieron en cuenta tres criterios para determinar el paso de tiempo:

- Criterio de estabilidad Δt_{max}.
- Δt_{user} dado por el usuario.
- Δt_{rec} de las trazas de entrada.

La figura 61 muestra el procedimiento de verificación para seleccionar el Δt final.

Ciclo de procesamiento de disparos. Una vez determinados los parámetros generales para el procesamiento de los datos, se comienza a leer las trazas disparo a disparo. En este punto el procesamiento realizado es propio de cada uno de los disparos.

Lee disparo. El archivo de trazas se lee secuencialmente. Para ello se empleó la rutina de SU `fvgettr` y se actualiza la estructura wavelet que almancena las trazas y la posición de la fuente. Es posible procesar algunos de los disparos de un dato mediante los parámetros `nshots`, `incshot` y `ishot`. En este punto el algoritmo lee y descarta los disparos que no se deben procesar.

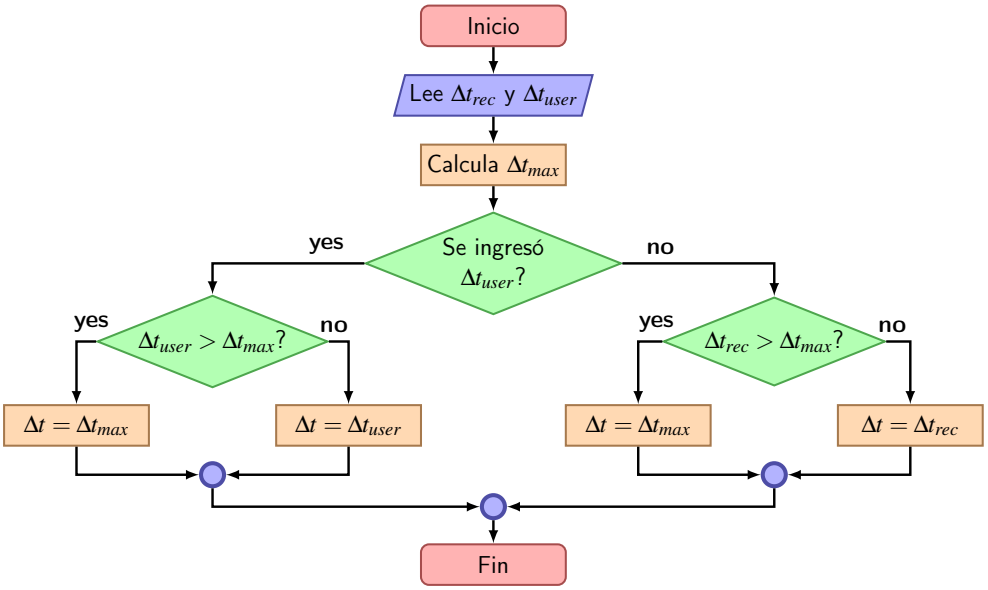


Figura 61. Diagrama de flujo de la selección de Δt

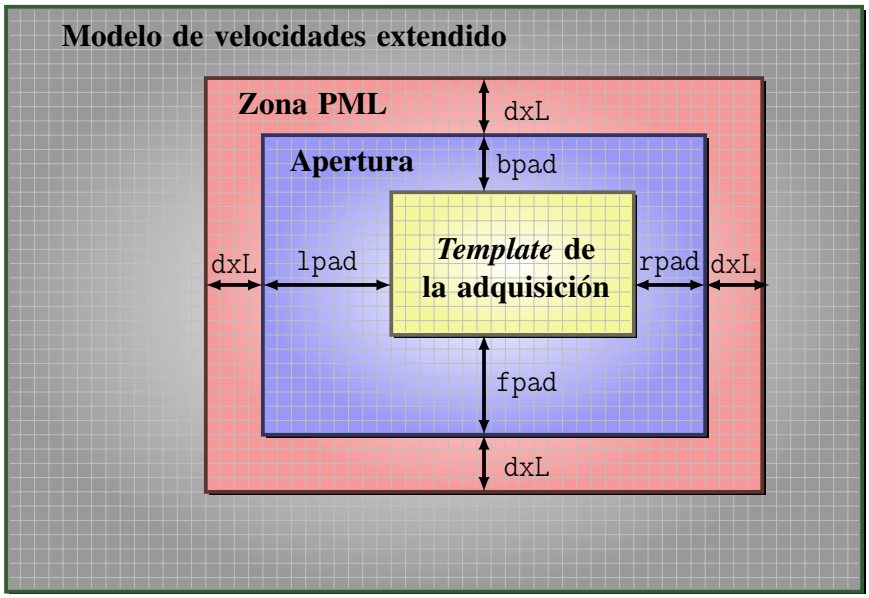


Figura 62. Vista superior del proceso de extracción del submodelo de velocidades.

Interpolación de trazas. Según el Δt determinado en la sección 2, se deben interpolar las trazas en el tiempo. La interpolación se hace mediante la rutina `ints8r` que hace parte del comando `suresamp` de SU.

Extracción del submodelo de velocidades.. El procesamiento de un disparo no se realiza sobre el modelo completo debido a que su aporte se concentra en los alrededores del template. Esto permite reducir los requerimientos de memoria y realizar el procesamiento sobre un submodelo extraido del modelo completo. Los parámetros `lpad`, `rpad`, `fpad`, `bpad`, `tpad` y `opad` permiten modificar el tamaño del submodelo en cada una de las dimensiones. La figura 62 muestra la vista superior de este proceso.

Adicionalmente, el submodelo se extiende `L` puntos en cada dirección creando una zona dedicada a la condición de frontera absorbente. En caso de que la apertura o el ancho de la zona PML sea tan grande que el modelo extendido no logre contener el submodelo deseado, entonces éste es reducido hasta los límites del modelo extendido.

Para llevar a cabo estas acciones, inicialmente se determina la extensión del template sobre las trazas del último disparo leído. Estos límites quedan guardados en las variables `rec_min_x`, `rec_max_x`, `rec_min_y`, `rec_max_y`, `rec_min_z` y `rec_max_z`. El cálculo de las coordenadas límite y el tamaño del submodelo está dado por:

$$\texttt{c\_x0} = \text{máx}\left(\texttt{rec\_min\_x} - \texttt{lpad} - \texttt{L} * \texttt{dx}, -\texttt{ilext} * \texttt{dx}\right) \tag{85}$$

$$\texttt{c\_y0} = \text{máx}\left(\texttt{rec\_min\_y} - \texttt{bpad} - \texttt{L} * \texttt{dy}, -\texttt{ibext} * \texttt{dy}\right) \tag{86}$$

$$\texttt{c\_z0} = \text{máx}\left(\texttt{rec\_min\_z} - \texttt{tpad} - \texttt{L} * \texttt{dz}, -\texttt{itext} * \texttt{dz}\right) \tag{87}$$

$$\texttt{c\_xf} = \text{mín}\left(\texttt{rec\_max\_x} + \texttt{rpad} + \texttt{L} * \texttt{dx}, -(\texttt{FNx} - \texttt{ilext} - 1) * \texttt{dx}\right) \tag{88}$$

$$\texttt{c\_yf} = \text{mín}\left(\texttt{rec\_max\_y} + \texttt{fpad} + \texttt{L} * \texttt{dy}, -(\texttt{FNy} - \texttt{ibext} - 1) * \texttt{dy}\right) \tag{89}$$

$$\texttt{c\_zf} = \text{mín}\left(\texttt{rec\_max\_z} + \texttt{opad} + \texttt{L} * \texttt{dz}, -(\texttt{FNz} - \texttt{itext} - 1) * \texttt{dz}\right) \tag{90}$$

$$\texttt{Nx} = \left[\frac{\texttt{c\_xf-cx\_0}}{\texttt{dx}}\right] + 1 \qquad \texttt{Ny} = \left[\frac{\texttt{c\_yf-cy\_0}}{\texttt{dy}}\right] + 1 \qquad \texttt{Nz} = \left[\frac{\texttt{c\_zf-cz\_0}}{\texttt{dz}}\right] + 1 \tag{91}$$

Llamado a la estrategia seleccionada. El llamado a la estrategia se hace dentro de la función `RTM_core`, que invoca el procedimiento establecido para cada una de las estrategias programadas. Dentro de estas rutinas, el sistema de coordenadas de referencia se mueve para que el origen coincida con el origen del submodelo extraído. Los resultados del procesamiento de un disparo aportan al resultado final teniendo en cuenta la ubicación del submodelo dentro del modelo completo.

Procesamiento post-disparo y final. Una vez se haya procesado el disparo, se realiza la condición de imagen y se aplican los filtros para eliminar artefactos de baja frecuencia. Este procedimiento aplica igualmente al finalizar el procesamiento de todos los disparos.

Condición de imagen

Inspirado en trabajos como Vivas and Pestana (2007); Vivas et al. (2009); Guitton et al. (2006), se implementaron varias formas de la condición de imagen:

0: Tradicional
$$R(x,y,z) = \sum_{\forall shots} \sum_{t=0}^{t_{max}} p_s(x,y,z,t) \times p_r(x,y,z,t). \quad (92)$$

1: Normalizada a la fuente en cada disparo
$$R(x,y,z) = \sum_{\forall shots} \frac{\sum_{t=0}^{t_{max}} p_s(x,y,z,t) \times p_r(x,y,z,t)}{\sum_{t=0}^{t_{max}} p_s^2(x,y,z,t)}. \quad (93)$$

2: Normalizada a los receptores en cada disparo
$$R(x,y,z) = \sum_{\forall shots} \frac{\sum_{t=0}^{t_{max}} p_s(x,y,z,t) \times p_r(x,y,z,t)}{\sum_{t=0}^{t_{max}} p_r^2(x,y,z,t)}. \quad (94)$$

3: Normalizada a la fuente
$$R(x,y,z) = \frac{\sum_{\forall shots} \sum_{t=0}^{t_{max}} p_s(x,y,z,t) \times p_r(x,y,z,t)}{\sum_{\forall shots} \sum_{t=0}^{t_{max}} p_s^2(x,y,z,t)}. \quad (95)$$

4: Normalizada a los receptores
$$R(x,y,z) = \frac{\sum_{\forall shots} \sum_{t=0}^{t_{max}} p_s(x,y,z,t) \times p_r(x,y,z,t)}{\sum_{\forall shots} \sum_{t=0}^{t_{max}} p_r^2(x,y,z,t)}. \quad (96)$$

Ejemplo de cálculo de los parámetros de entrada y uso de Memoria

En esta sección se presentará detalladamente el cálculo de los parámetros de ejecución para un dato de entrada de ejemplo. La tabla 22 presenta las características de los datos de entrada.

El modelo de velocidades es leído en la parte inicial del programa y ahí mismo se verifica la coherencia entre el tamaño del archivo leído y las dimensiones reportadas en `userFNx`, `userFNy`, `userFNz`. Para

Tabla 22
Parámetros de entrada de ejemplo

Modelo de velocidades	
N_x, N_y, N_z [puntos]	$(100, 100, 100)$
$\Delta_x, \Delta_y, \Delta_z$ [m]	$(10, 10, 10)$
V_{min} [m/s]	1500
V_{max} [m/s]	4700
Extensión del modelo	
left [m]	320
right [m]	320
back [m]	320
front [m]	320
top [m]	400
bottom [m]	320
Parámetros de ejecución	
`pplo`	10
`Lpml` [puntos]	16
`dt`(usuario) [s]	0.002
`fq` [Hz]	20
`order`	8
`abc`	1, 1, 1, 1, 1, 1

Trazas sísmicas	
tmax [s]	1.2
dtrec [s]	0.0002
nshots	25
Trazas por disparo	676
Rango receptores disparo 1	
x:[left-right]	[50 - 550]
y:[back-front]	[50 - 550]
z:[top-bottom]	[0 - 0]
Apertura disparo 1	
left	100
right	100
back	100
front	100
top	400
bottom	1000

este caso el tamaño del archivo es $100 \times 100 \times 100 \times 4$ [Bytes].

Posteriormente el modelo es extendido según la ecuaciones 78, 79 y 80, dando como resultado:

$$\texttt{userFNx\_ext} = 100 + \left\lfloor \frac{320}{10} \right\rfloor + \left\lfloor \frac{320}{10} \right\rfloor = 164 \tag{97}$$

$$\texttt{userFNy\_ext} = 100 + \left\lfloor \frac{320}{10} \right\rfloor + \left\lfloor \frac{320}{10} \right\rfloor = 164 \tag{98}$$

$$\texttt{userFNz\_ext} = 100 + \left\lfloor \frac{400}{10} \right\rfloor + \left\lfloor \frac{320}{10} \right\rfloor = 172 \tag{99}$$

Ahora se remuestrea el modelo extendido para garantizar la cantidad de puntos por longitud de onda (`pplo`) dada por el usuario. El valor de Δs, según la ecuación 81 está dado por:

$$\Delta s = \frac{V_{min}}{\texttt{pplo} * f_q} = \frac{1500}{10 * 20} = 7.5[m] \tag{100}$$

Los valores definitivos de la resolución del modelo serán divisiones enteras del valor de entrada, entonces siguiendo con las ecuaciones 82 y 83:

$$\texttt{factorx} = \left\lceil \frac{10}{7.5} \right\rceil = 2 \qquad \texttt{factory} = \left\lceil \frac{10}{7.5} \right\rceil = 2 \qquad \texttt{factorz} = \left\lceil \frac{10}{7.5} \right\rceil = 2 \tag{101}$$

$$\texttt{dx} = \frac{10}{2} = 5[m] \qquad \texttt{dy} = \frac{10}{2} = 5[m] \qquad \texttt{dz} = \frac{10}{2} = 5[m]. \tag{102}$$

Finalmente el nuevo tamaño del modelo remuestreado será según la ecuación 84:

$$\texttt{FNx} = (\texttt{userFNx\_ext} - 1)\texttt{factorx} + 1 = (164-1)*2+1 = 327 \tag{103}$$

$$\texttt{FNy} = (\texttt{userFNy\_ext} - 1)\texttt{factory} + 1 = (164-1)*2+1 = 327 \tag{104}$$

$$\texttt{FNz} = (\texttt{userFNz\_ext} - 1)\texttt{factorz} + 1 = (172-1)*2+1 = 343. \tag{105}$$

Ahora para elegir el Δt que se va a usar, se tienen: $\Delta t_{user} = 0.002$, $\Delta t_{rec} = 0.0002$ y el criterio de estabilidad

$$\Delta t_{max} = \frac{2\,\text{mín}(\Delta x, \Delta y, \Delta z)}{\sqrt{3}V_{max}\sqrt{-C_0 + \sum_{l=1}^{N/2} C_l(-1)^{l+1}}} = \frac{2\times 5}{4700\sqrt{3}\sqrt{6.5015873}} = 0.00048176. \tag{106}$$

Siguiendo el diagrama de flujo de la Figura 61, el paso de tiempo elegido es

$$\Delta t = \Delta t_{max} = 0.00048176[s] \tag{107}$$

y el proceso de modelado tendrá

$$\texttt{itmax} = \left\lfloor \frac{tmax}{\Delta t} \right\rfloor = \left\lfloor \frac{1.2}{0.00048176} \right\rfloor = 2490[pasos] \tag{108}$$

Estrategias para procesar cada disparo.

La base del procesamiento de cada disparo es la estrategia del manejo de la memoria mediante la cual se administra este recurso para llevar a cabo el algoritmo. La modularidad de la implementación ha segmentado el algoritmo en varios *kernels* que se ejecutan en la GPU. Los principales *kernels* se listan a continuación:

Kernel de propagación: Evalúa mediante diferencias finitas la ecuación 19. Dado que esta ecuación es simétrica para los términos del campo en el tiempo futuro(p^{k+1}) y el tiempo pasado (p_{k-1}), este mismo *kernel* se emplea en la propagación y la retropropagación de los campos.

Kernels de PML: Adiciona los términos adicionales de la ecuación de onda que aparecen en la ecuación 30. Existe un *kernel* por cada una de las seis caras del espacio representado.

Kernel para la condición de imagen: Según la condición de imagen elegida por el usuario, este *kernel* calcula la condición de imagen acumulando su valor en la variable `r`.

Kernel para adicionar del término de la fuente: Este *kernel* permite adicionar el término de la fuente para el modelado y adicionar la información de los receptores en la retropropagación. En ambos casos recibe la estructura `wavelet`.

Kernel para inicializar las variables PML: Inicializa las variables ψ_x, ψ_y, ψ_z, ζ_x, ζ_y, ζ_z, a_x, a_y, a_z, b_x, b_y y b_z de las ecuaciones 26 y 27.

Kernels para almacenar y recuperar la frontera: Se emplean durante la ejecución de la estrategia 2 dado que ésta requiere almacenar la frontera del campo durante la propagación del campo de la fuente y restablecer la misma durante el proceso de reconstrucción hacia atrás.

Kernels para fronteras aleatorias: Son funciones para la generación de los valores aleatorios en las fronteras del modelo y para la generación de las semillas del generador de números aleatorios. Estos *kernels* son empleados exclusivamente en la estrategia 3.

A continuación se dan algunos detalles de cada una de las estrategias.

Tabla 23
Ejemplo de mapeo del macro `d_s(n,i,j,k)` para `ks_store=5`

La expresión...	Direcciona a...
`d_s(0,...)`	`d_s[0*Nx*Ny*Nz+...]`
`d_s(1,...)`	`d_s[1*Nx*Ny*Nz+...]`
`d_s(2,...)`	`d_aux_s1[...]`
`d_s(3,...)`	`d_aux_s2[...]`
`d_s(4,...)`	`d_aux_s1[...]`
`d_s(5,...)`	`d_s[2*Nx*Ny*Nz+...]`
`d_s(6,...)`	`d_s[3*Nx*Ny*Nz+...]`
`d_s(7,...)`	`d_aux_s2[...]`
`d_s(8,...)`	`d_aux_s1[...]`
`d_s(9,...)`	`d_aux_s2[...]`

Estrategia 1. La estrategia usa la memoria disponible en la GPU para almacenar puntos de control que permitan reanudar el modelado de la fuente cuando sea necesario. Para un punto de control $it = it_{chk}$ se deben almacenar los valores del campo para los tiempos it_{chk} y $it_{chk}+1$, y las dos variables auxiliares de PML ψ y ζ para el tiempo t_{chk}.

Para el campo de la fuente se creó un macro `d_s(n,i,j,k)` que permite emular el acceso a todo el campo mapeando cada paso de tiempo a otras variables con memoria física reservada. El cuadro de texto muestra el código fuente del macro y la Tabla 23 un ejemplo de mapeo para un `ks_store=5`. Las variables `d_aux_s1` y `d_aux_s2` almacenan pasos de tiempo impares y pares del campo respectivamente, y la variable `d_s` almacena todos los puntos de control del campo de la fuente.

La figura 63 muestra el diagrama de flujo de la estrategia. Inicialmente se reserva la memoria requerida para la ejecución del algoritmo y se inicializan variables como los campos ψ y ζ del PML y los valores

iniciales de campo en el modelado. Posteriormente se hace el modelado de la fuente con la variable `its` desde el paso de tiempo `0` hasta `itmax-1`. El modelado aplica el paso de tiempo, que incluye el PML, suma el término de la fuente y finalmente guarda el campo calculado en disco, si el usuario así lo solicita. En los puntos donde `its` es un punto de control, se hace una copia de las variables PML. El campo de la fuente es guardado automáticamente en los puntos de control gracias al macro `d_s`.

```
#define d_s(n,i,j,k)  \
    (((n)%ks_store==0)?d_s[Nx*Ny*Nz*(((n)*2)/ks_store  )+Ny*Nz*(i)+Nz*(j)+(k)]: \
    (((n)%ks_store==1)?d_s[Nx*Ny*Nz*(((n)*2)/ks_store+1)+Ny*Nz*(i)+Nz*(j)+(k)]: \
    (((n)%2==0)?d_aux_s1[Ny*Nz*(i)+Nz*(j)+(k)]:d_aux_s2[Ny*Nz*(i)+Nz*(j)+(k)])))
```

Macro para emular un espacio de memoria completo para el campo de la fuente

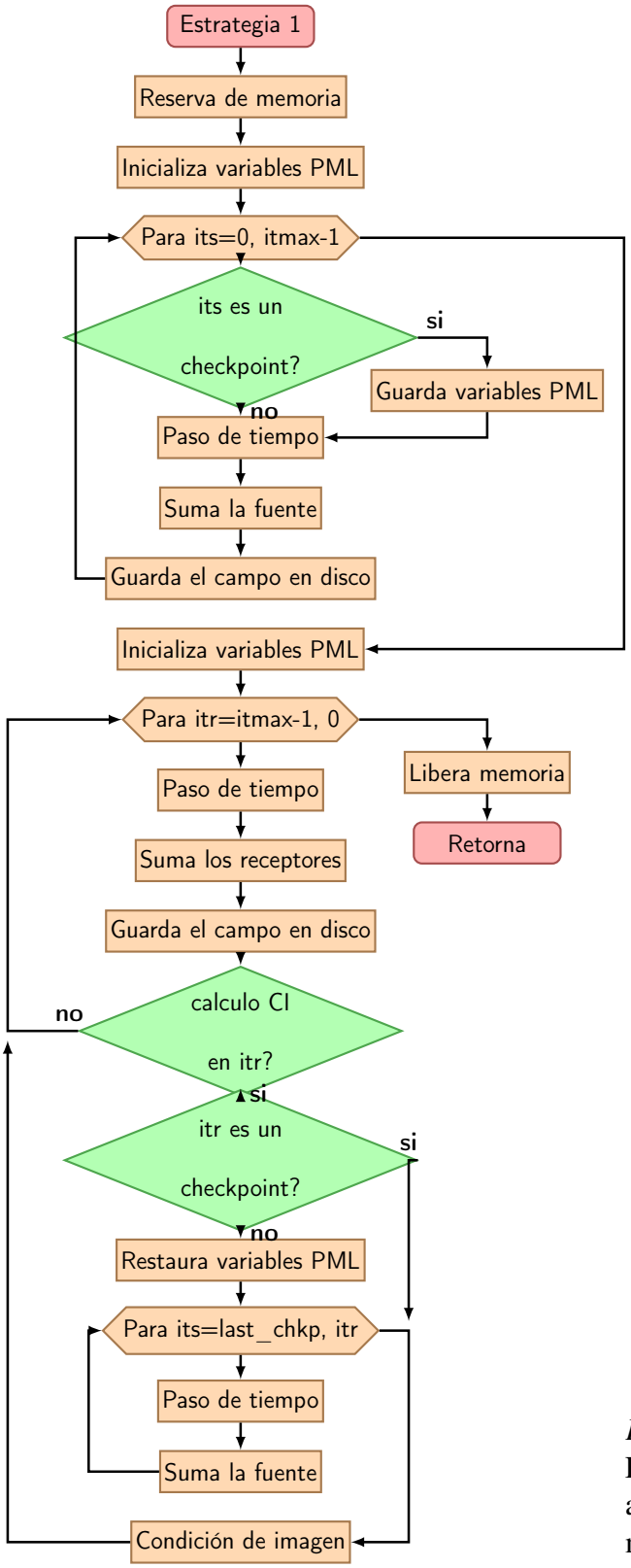


Figura 63. Diagrama de flujo de la estrategia 1. Esta estrategia almacena el campo en puntos de control para posteriormente reconstruirlo.

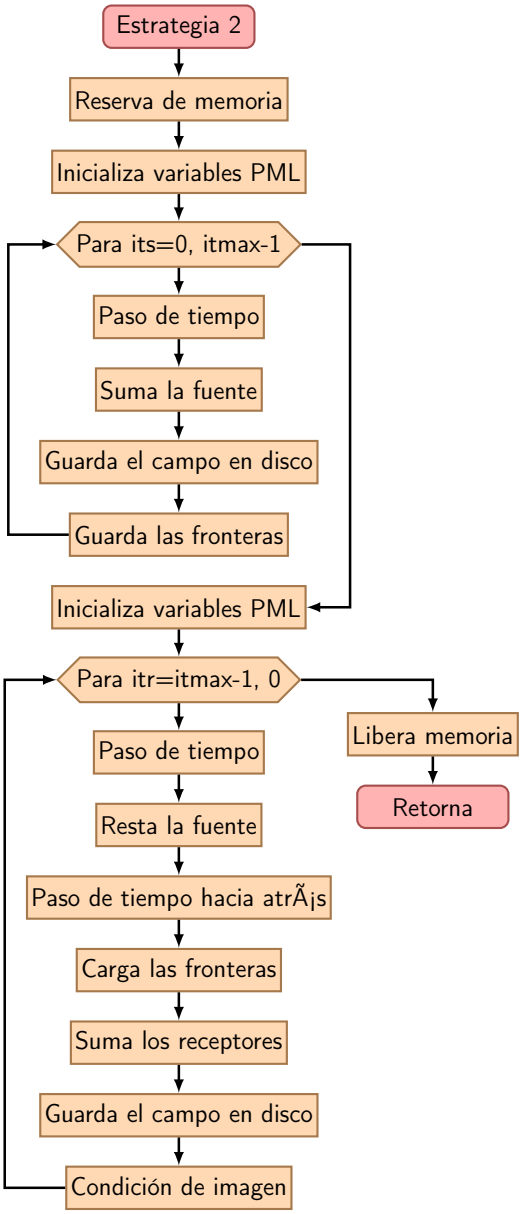


Figura 64. Diagrama de flujo de la estrategia 2. Esta estrategia reconstruye el campo hacia atrás, para lo cual almacena los valores del mismo en la frontera.

En la segunda parte de la implementación, se realiza la retropropagación del campo de los receptores y se aplica la condición de imagen. Primero se inicializaron las variables PML del campo de los receptores y se inicia el ciclo con la variable `itr` desde `itmax-1` a `0`. Al igual que en el campo de la fuente, se aplican los *kernel* para avanzar(retroceder) un paso de tiempo, se suma la fuente y se almacena en disco, en caso de que el usuario lo requiera. En caso de que `itr` corresponda a un punto para aplicar la condición de imagen, se debe contar con el campo de la fuente en el mismo paso de tiempo. Si `itr` coincide con un punto de control, se calcula directamente la condición de imagen, sino, se debe iniciar un subciclo para repetir el modelado desde el último punto de control.

Al final del proceso se libera la memoria de la CPU y de la GPU empleada durante la estrategia. El mapa de reflectividad parcial generado al procesar el disparo retorna en la variable `r`.

Estrategia 2. Esta estrategia propaga y retropropaga el campo de la fuente para lo cual almacena la frontera del campo en cada uno de los pasos de tiempo durante el modelado. Para almacenar la frontera se empleó la estructura `boundary` que agrupa un arreglo para cada una de las caras del cubo.

El macro `d_s(n,i,j,k)` es una versión simplificada del utilizado en la estrategia 1 donde solo se utilizan las variables `d_aux_s1` y `d_aux_s2` para direccionar los pasos de tiempo del campo pares e impares respectivamente. El código del macro se puede apreciar en el cuadro de código.

```
1 #define d_s(n,i,j,k)  \
2     (((n)%2==0)?d_aux_s1[Ny*Nz*(i)+Nz*(j)+(k)]:d_aux_s2[Ny*Nz*(i)+Nz*(j)+(k)]))
```

Macro para emular un espacio de memoria completo para el campo de la fuente

La figura 64 muestra el diagrama de flujo de la rutina de la estrategia. Al igual que en la estrategia 1, se inicia reservando e inicializando las variables requeridas para el modelado del campo de la fuente. Cada paso en el modelado implica avanzar un paso de tiempo mediante el *kernel* de propagación, los *kernels* de PML y la adición del término de la fuente. Posteriormente, si el usuario habilitó la bandera `smovie`, el algoritmo guarda en disco el campo de onda y las fronteras en la estructura `boundary` mediante el *kernel* `save_boundaries_gpu`.

Al finalizar el modelado, se inicializan las variables PML para los receptores y se comienza el ciclo de retropropagación. El campo de los receptores se retropropaga de la misma forma aplicando los *kernels* de propagación, los *kernels* PML y la adición de los receptores. La retropropagación del campo de la fuente se realiza utilizando únicamente el kernel de propagación y recuperando las fronteras con el *kernel* `load_boundaries_gpu`.

Al retropropagar los dos campos, se puede almacenar en disco el campo de los receptores, si el usuario activó la bandera `rmovie`; y el campo retropropagado de la fuente, si el usuario activó la bandera `sbackmovie`. Así mismo se calcula la condición de imagen y se aporta a la respuesta final en la variable `r`. Al final del proceso se libera la memoria reservada en la etapa inicial de la estrategia.

Estrategia 3. La estrategia 3, al igual que la estrategia 2, propaga y retropropaga el campo de la fuente, pero sin almacenar las fronteras debido a que no usa PML. En cambio emplea una región de frontera de velocidades aleatorias que genera un patrón de reflexiones incoherentes con el campo de los receptores.

El macro `d_s(n,i,j,k)` es el mismo que se empleó en la estrategia 2 que solo emplea dos arreglos:

`d_aux_s1` y `d_aux_s2` para direccionar los pasos de tiempo del campo.

La Figura 65 muestra el diagrama de flujo de la estrategia. En los primeros pasos, además de declarar e inicializar las variables de modelado y PML, se inicializan las variables relacionadas con el proceso de generación de números aleatorios. Posteriormente en el ciclo de modelado del campo de la fuente, se tiene el avance de un paso de tiempo sin PML, la adición del término de la fuente y la posibilidad de almacenar el campo en el disco, si el usuario así lo establece. Iniciando este ciclo se incluye la variación de la frontera aleatoria que periódicamente, cada `ks_rand` pasos de tiempo, cambia el valor de la velocidad mediante el *kernel* `randomize_vmodel_boundaries_gpu`.

Posteriormente inicia el ciclo de retropropagación del campo de los receptores en dirección inversa. El campo de los receptores se realiza de la misma forma que en las estrategias anteriores mediante los *kernel* de propagación, PML y sumando el aporte de las trazas. El campo de la fuente se retropropaga con los *kernels* de propagación y restando el aporte de la fuente. Durante este proceso se invierte el proceso de generación de valores aleatorios recuperando los modelos de velocidades empleados durante el modelado. Finalmente se libera la memoria y se devuelve el resultado en la variable `r`

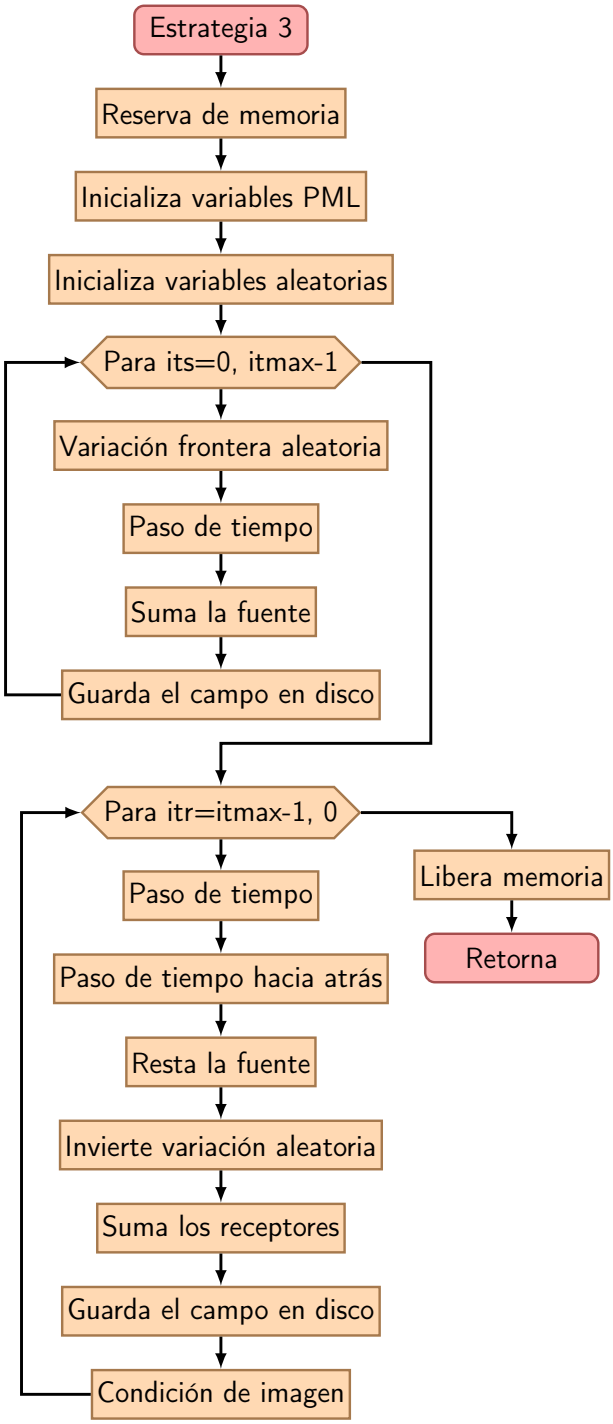


Figura 65. Diagrama de flujo de la estrategia 3. Esta estrategia sustituye el PML en la propagación del campo de la fuente por una frontera aleatoria que permite reconstruir el campo sin guardar la frontera.