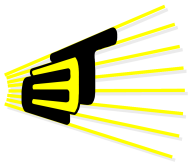
IMPLEMENTACIÓN DE UN ALGORITMO DE INVERSIÓN DE ONDA COMPLETA (FWI) 2D CON ANISOTROPÍA VTI UTILIZANDO UN CLUSTER DE GPUS.

PETTER ALEXIS MAYORGA TRIANA
DANIEL ALEJANDRO VEGA NIEVES



CPS | RESEARCH GROUP

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

IMPLEMENTACIÓN DE UN ALGORITMO DE INVERSIÓN DE ONDA COMPLETA (FWI) 2D CON ANISOTROPÍA VTI UTILIZANDO UN CLUSTER DE GPUS.

PETTER ALEXIS MAYORGA TRIANA
DANIEL ALEJANDRO VEGA NIEVES

Trabajo de grado presentado como requisito para optar al título de

Ingeniero Electrónico

Director
Ana Beatriz Ramírez Silva
Ph.D. en Ingeniera Eléctrica

Codirector
Sergio Alberto Abreo Carrillo
Dr.Ing. en Ingeniera Eléctrica, Electrónica y Gestión & Desarrollo

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

AGRADECIMIENTOS

Quisiera expresar lo agradecido que estoy con mis padres, pues es gracias a ellos que complete este logro. Las palabras no alcanzan para expresar lo agradecido que estoy con ellos pues fueron ellos quienes siempre me empujaron a culminar cada una de mis metas. Gracias a ellos creció en mí una fuerte determinación y pasión hacia la ingeniería porque fueron ellos los que me enseñaron que en el conocimiento esta la verdadera satisfacción de la vida. Hoy este pequeño logro conforma una felicidad inmensa para mi y se que en ellos también. Y esa felicidad que crece dentro de ellos es por el amor que sienten hacia mí, el mismo que les dio la vitalidad necesaria para superar las trampas que les puso la vida y ayudarme a llegar hasta acá. Gracias.

Daniel Vega Nieves

Quiero agradecer a mis padres Pedro Mayorga y Carmen Triana, quienes siempre me han brindado su incondicional apoyo en las decisiones que he tomado. Por los sacrificios que han hecho para lograr que hoy yo cumpla mis metas. Por el impulso que me brindaron en los momentos más difíciles de mi carrera. Por ayudarme a vencer esos obstáculos. Por inculcarme el valor que tiene la educación y el saber. A las personas cercanas a mi que me brindaron su mano cuando la necesité. A ustedes debo este logro y con ustedes lo comparto. Gracias.

Petter Mayorga Triana

Agradecemos fervientemente a Fabián Sanchez, nuestro guía y tutor durante el desarrollo de gran parte del trabajo. A nuestra director y codirector, los decentes Ana Ramírez y Sergio A. Abreo por su apoyo y esfuerzo académico en la elaboración de nuestro trabajo de grado. A todos nuestros compañeros y personas allegadas que de una forma u otra siempre nos apoyaron durante el camino, muchas gracias.

Este trabajo cuenta con el apoyo de la compañía petrolera de Colombia, ECOPETROL y COLCIENCIAS como parte del proyecto de investigación 0266 de 2013 y el acuerdo 004 de 2014. Los autores agradecen el apoyo del grupo de investigación CPS de la Universidad Industrial de Santander y el Instituto Colombiano del Petróleo, ICP.

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE CUADROS

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

TÍTULO: Implementación de un algoritmo de inversión de onda completa (FWI) 2D con anisotropía VTI utilizando un cluster de GPUs. *

AUTORES: Petter Alexis Mayorga Triana **
Daniel Alejandro Vega Nieves **

PALABRAS CLAVE: Inversión de onda completa, FWI, anisotropía VTI, GPU, Checkpoint.

DESCRIPCIÓN

Este proyecto busca desarrollar e implementar estrategias computacionales para desarrollar la Inversión de onda completa o FWI por sus siglas en inglés *Full Wave Inversion* en arquitecturas de computo en paralelo o GPU del inglés *Graphic Processing Units*. Estas estrategias consisten en realizar un mejor manejo de la memoria, con el fin de que esta memoria no sea una limitante para modelos de datos muy grandes. Las estrategias que se van a implementar son dos, la primera se denomina *checkpointing method* o método del punto de control y consiste en segmentar la inversión de onda completa con el fin de utilizar los datos de una propagación parcial para hacer calculos parciales, haciendo varias propagaciones parciales y al final sumar los calculos parciales para obtener la inversión de onda completa en su totalidad. La segunda estrategia llamada *hidden latency method* o método de latencia oculta, consiste en aprovechar la memoria del *host*, haciendo transferencias de datos de manera asíncrona a través de código concurrente, ocultando así las latencias de dichas transferencias de datos. Ambos métodos se pueden analizar y comparar con la estrategia normal o tradicional, que consiste en desarrollar la inversión de onda completa manteniendo los datos en la *RAM* de la GPU.

*Trabajo de investigación.

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Directora: Ana Beatriz Ramírez Silva, PhD. en Ingeniera Eléctrica.

ABSTRACT

TITLE: Implementation of a Full Wave Inversion (FWI) 2D with VTI anisotropy algorithm using a cluster of GPUs. *

AUTHORS: Petter Alexis Mayorga Triana **
Daniel Alejandro Vega Nieves **

KEYWORDS: Full Wave Inversion, VTI, GPU, Checkpoint.

DESCRIPTION:

This project seeks to develop and implement computational strategies to develop the full wave inversion in parallel architectures (GPU). These strategies consist in a better management of memory, in order to make the memory a non-limitation parameter for very large data. The strategies implemented are the checkpointing method which is a segmentation of the data used in the FWI. The second strategy, hidden latency, wants to take advantage of the memory in the host making data transfers asynchronously through concurrent code, thus hiding the latencies of data transfer. Both methods can be compared with the traditional strategy which consists of developing the full wave inversion while keeping the data in the RAM of the GPU.

*Bachelor Thesis

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Ana Beatriz Ramírez Silva, PhD. en Ingeniera Eléctrica.

1. INTRODUCCIÓN

La obtención de imágenes sísmicas confiables y de alta resolución es de gran relevancia para la detección de hidrocarburos en la industria petrolera. La Inversión de Onda Completa (FWI por sus siglas en inglés *Full Waveform Inversion*) es un procedimiento que busca obtener las propiedades del subsuelo a partir de un conjunto de datos geofísicos[28]. Dado un modelo inicial conformado por determinados parámetros tales como la velocidad, la densidad, etc.., se computan dichos datos sísmicos mediante la solución de la ecuación de onda. Luego el modelo se actualiza de forma iterativa con el fin de reducir el error entre los datos observados y modelados. Este procedimiento se repite de manera iterativa hasta obtener un error suficientemente pequeño o se cumpla un número de iteraciones.

El medio a trabajar tiene propiedades que se deben tener en cuenta para mejorar la resolución de la imagen sísmica. Una de estas propiedades es la anisotropía sísmica que se define como la variación de la velocidad con la dirección de la propagación de onda [26]. Esta propiedad se puede encontrar en ciertas rocas con gran cantidad de fracturas y estratos delgados de sedimentos[27]. Según la orientación del eje de simetría, la anisotropía se puede clasificar en isotropía transversal vertical, (VTI por sus siglas en inglés *Vertical Transverse Isotropy*), isotropía transversal horizontal (HTI por sus siglas en inglés *Horizontal Transverse Isotropy*) e isotropía transversal inclinada (TTI por sus siglas en inglés *Tilted Transverse Isotropy*[9]).

La FWI requiere una función de costo, una ecuación de onda para encontrar el campo de presión propagado hacia adelante y hacia atrás y un modelo de velocidad inicial, para estimar un modelo de velocidades en forma iterativa. A pesar de las ventajas de la FWI, el método es costoso computacionalmente. El punto de partida del método también es una restricción. El método requiere de una buena predicción del modelo inicial para lograr convergencia y así lograr reducir la brecha entre el modelo calculado y el modelo real.

En la actualidad, la FWI demanda una gran cantidad de recursos computacionales. Utilizar este método de inversión toma mayor tiempo de ejecución en arquitecturas convencionales tales como las unidades de procesamiento central, mejor conocida por sus siglas en inglés *central processing unit* (CPU), ya que su ejecución es secuencial y esto hace lento el cálculo de operaciones. Sin embargo, gracias al desarrollo de arquitecturas en paralelo como las unidades de procesamiento gráfico o GPU (del inglés *graphic processing units*) la implementación de la FWI resulta viable gracias a su velocidad de procesamiento [33]. Por ende, este proyecto de investigación busca implementar la FWI 2D con anisotropía VTI sobre un cluster de GPUs, abordando el costo computacional que requiere la FWI, específicamente la memoria limitada de las GPUs.

El texto da una descripción teórica de la FWI y la manera en que esta se implementa para desarrollar el algoritmo. Se plantea el problema y la solución que se da a partir de estrategias computacionales desarrolladas que son *checkpointing method* y *Hidden latency method*. Finalmente se muestran los resultados obtenidos y las conclusiones que se tienen del análisis de dichos resultados.

2. MARCO TEÓRICO

2.1 ECUACIÓN DE ONDA ACÚSTICA ANISOTRÓPICA Y CPML

El aspecto clave de la técnica de inversión de onda completa (FWI) es modelar con precisión los campos de onda registrados en la superficie para representar la cinemática[1], y, en cierta medida, la dinámica[2] de todas las ondas durante la inversión iterativa hacia el modelo final de tierra.

La ecuación de onda pseudo-acústica usada para medios con anisotropía VTI 3D se puede, según Plessix [21], expresar como

$$\begin{aligned} \frac{\delta^2 P}{\delta t^2} &= c_x \frac{\delta^2 P}{\delta x^2} + c_y \frac{\delta^2 P}{\delta y^2} + c_z \frac{\delta^2 R}{\delta z^2}, \\ \frac{\delta^2 R}{\delta t^2} &= d_x \frac{\delta^2 P}{\delta x^2} + d_y \frac{\delta^2 P}{\delta y^2} + d_z \frac{\delta^2 R}{\delta z^2}, \end{aligned} \tag{1}$$

siendo P y R los campos de onda pseudo-acústicos y $c_i, d_i (i \in x, y, z)$ coeficientes de velocidad dados por

$$\begin{aligned} c_x = c_y &= \nu_v^2 (1 + 2\epsilon), \\ d_x = d_y &= \nu_v^2 (1 + 2\delta), \\ c_z = d_z &= \nu_v^2, \end{aligned} \tag{2}$$

donde ν_v es la velocidad vertical y ϵ, δ son los parámetros de anisotropía de Thomsen. Thomsen introdujo los parámetros anisótropos adimensionales ϵ y δ que rigen la propagación de ondas P en medios de anisotropía débil, donde las variaciones de la velocidad no son mayores al 20 % [9].

[1] Cinemática hace referencia al análisis espacial y temporal del movimiento del frente de onda y su evolución a través del tiempo, sin tener en cuenta las causas que lo producen.

[2] La dinámica hace un análisis de los parámetros que producen perturbaciones o cambios físicos en un modelo.

Los coeficientes de velocidad c_x, c_z y d_x representan las tres velocidades de la onda P, donde c_x es la velocidad en la dirección normal al eje de simetría, c_z la velocidad en la dirección del eje de simetría y d_x es la velocidad NMO.
La distancia entre una fuente sísmica y un receptor o geófono es llamado el *offset*. La velocidad NMO del inglés *normal moveout*, describe el efecto que tiene este *offset* en el tiempo de llegada de un reflejo de onda al receptor. El efecto se da en forma de un aumento de tiempo con el *offset*. La relación entre el tiempo de llegada y el *offset* es hiperbólica y es el principal criterio que se utiliza en geofísica para decidir si un evento es un reflejo o no [7].

El set de ecuaciones usadas para el caso 2D son

$$\begin{aligned} \frac{\delta^2 P}{\delta t^2} &= c_x \frac{\delta^2 P}{\delta x^2} + c_z \frac{\delta^2 R}{\delta z^2}, \\ \frac{\delta^2 R}{\delta t^2} &= d_x \frac{\delta^2 P}{\delta x^2} + d_z \frac{\delta^2 R}{\delta z^2}. \end{aligned} \tag{3}$$

Las condiciones de frontera deben tenerse en cuenta en la solución de la ecuación de onda acústica utilizando alguna técnica de análisis numérico. Estas condiciones de frontera son utilizadas para minimizar los reflejos artificiales al interior del área de interés. Se han propuesto varios métodos para incluir las condiciones de frontera en la solución de la ecuación de onda acústica, entre ellos PML del inglés *Perfectly Matched Layer* y CPML del inglés *Convolutional Perfectly Matched Layer* [20].
La ecuación de onda acústica anisotrópica incluyendo CPML requiere de dos variables auxiliares adicionales en cada dimensión espacial. Modificando la ecuación 3 tenemos que

$$\begin{aligned} \frac{\delta^2 P}{\delta t^2} &= c_x \frac{\delta^2 P}{\delta x^2} + c_z \frac{\delta^2 R}{\delta z^2} + c_x \left(\frac{\delta \psi_{p,x}}{\delta x} + \zeta_{p,x} \right) + c_z \left(\frac{\delta \psi_{r,z}}{\delta z} + \zeta_{r,z} \right), \\ \frac{\delta^2 R}{\delta t^2} &= d_x \frac{\delta^2 P}{\delta x^2} + d_z \frac{\delta^2 R}{\delta z^2} + d_x \left(\frac{\delta \psi_{p,x}}{\delta x} + \zeta_{p,x} \right) + d_z \left(\frac{\delta \psi_{r,z}}{\delta z} + \zeta_{r,z} \right), \end{aligned} \tag{4}$$

donde ψ y ζ son variables auxiliares con evolución en tiempo dado por

$$\begin{aligned} \psi_i^n &= a_i \psi_i^{n-1} + b_i \left(\frac{\delta P}{\delta i} \right)^n, \\ \zeta_i^n &= a_i \zeta_i^{n-1} + b_i \left[\left(\frac{\delta^2 P}{\delta i^2} \right)^n + \left(\frac{\delta \psi_i}{\delta i} \right)^n \right]; \end{aligned} \tag{5}$$

donde n denota el nivel de tiempo actual. Los parámetros a_i y b_i se pueden obtener a partir de la definición dada en [6] mostrada a continuación:

$$d_x = d_0 V_{max} \left(\frac{f(x)}{Lx} \right)^2, \tag{6}$$

$$\alpha_x = \pi f \left(\frac{Lx - f(x)}{Lx} \right), \tag{7}$$

$$b_x = \exp^{-(d_x + \alpha_x)^{dt}}, \tag{8}$$

$$a_x = \frac{d_x}{d_x - \alpha_x}(b_x - 1). \tag{9}$$

Se calcularan a modo ejemplo los parámetros anteriores para la dimensión x.

$$\begin{aligned} R &= 0,001, \\ L_x &= CPMLimit * \Delta h, \\ d_0 &= \frac{-3}{2L_x} \log(R). \end{aligned} \tag{10}$$

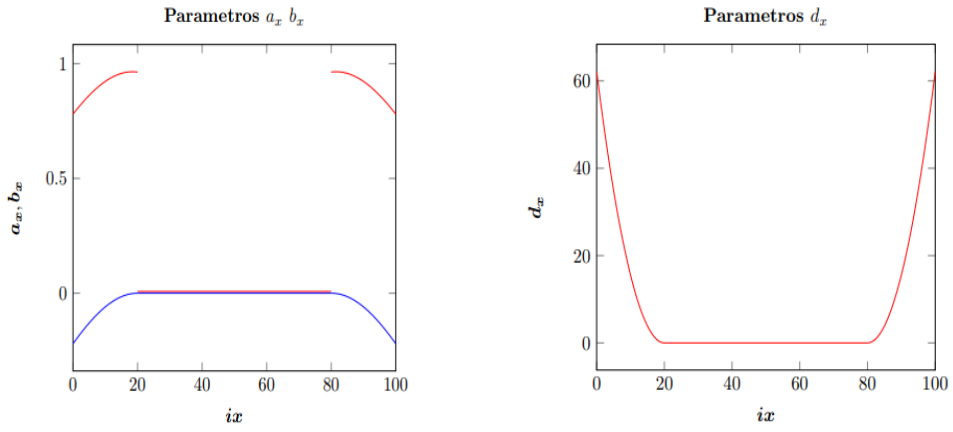
Estos tres valores, son constantes para cualquier posición y momento de la propagación, Δh, es el valor del paso espacial, y $CPMLimit$ es la cantidad de puntos sobre la cual se desea realizar la atenuación. Así, L_x es el ancho de la región de CPML.

Se define un vector X como:

$$X = \begin{cases} L_x : \Delta h : 0, & x \in (0, CPML], \\ 0, & x \in (CPML, N_x - CPML), \\ 0 : \Delta h : L_x, & x \in (N_x - CPML, N_x) \end{cases} \tag{11}$$

Las figuras 1(a) y 1(b) muestra los parámetros a_x, b_x y d_x como función de x.
Al igual que el vector X, estos parámetros se definen como 0 para regiones donde no se aplique atenuación CPML, es decir, $20 < x < 80$, lo que implica que las variables ψ_x^n y ζ_x^n también son cero, y por lo tanto la actualización para el campo escalar está dada por la solución de la ecuación de onda de campo 3. Por otra parte, las ecuaciones de onda con CPML dadas en 4 , sólo serán validas en las fronteras atenuantes, es decir, cuando $x < 20$ y $x > 80$ [22]. La forma como se definen a_x y b_x, se extrapola a las dimensiones restantes(z).
Para el cálculo de a y b en las dimensiones restantes (z) se hace el mismo análisis.

Figura 1: Parámetros a_x, b_x y d_x en función de x.



(a) Valores de a_x en azul y b_x en rojo asumiendo $N_x = 100$, $CPMLimit = 20$, $dt = 4m$.

(b) Valores de d_x asumiendo $Nx = 100$, $CPMLimit = 20$, $dt = 4m$.

2.2 DISCRETIZACIÓN

Para hacer la discretización de las ecuaciones utilizadas para el desarrollo del algoritmo se usa la técnica de diferencias finitas de octavo orden en espacio y segundo orden en tiempo. Para definir la localización espacial $[columna, fila]$ se usara la siguiente notación, teniendo en cuenta que las filas indican la profundidad.

$$P_{posición} = \begin{cases} Posición\ \ central = [i, j] \\ Posición\ \ izquierda = [i-1, j] \\ Posición\ \ derecha = [i+1, j] \\ Posición\ \ arriba = [i, j-1] \\ Posición\ \ abajo = [i, j+1] \end{cases} \tag{12}$$

Para la implementación de las ecuaciones de onda es necesaria la discretización de las mismas. Siguiendo la notación anterior y con el método de diferencias finitas de octavo orden en espacio y segundo orden en tiempo, la ecuación 3 se discretiza como se puede ver en la ecuación 59 del anexo 8.

Para la implementación hecha en este trabajo, el paso espacial Δx y Δz son iguales, por esto se reemplazan por Δh en las siguientes ecuaciones para mantener un solo coeficiente de paso espacial. Δt es el paso de tiempo.

La ecuación 59 contiene coeficientes propios del método de diferencias finitas en octavo orden espacial y en segundo orden temporal. Los factores $P^n_{i,j}$ y $P^{n-1}_{i,j}$ son cero para las dos primeras iteraciones pues son valores en tiempos anteriores y estos se suponen como cero.

Para la implementación se divide cada ecuación en dos partes, la primera calcula las derivadas del campo espacialmente y se guardan en una variable temporal P_{temp}. Luego, sumando los valores del campo presente y pasado ($P^n_{i,j}$ y $P^{n-1}_{i,j}$) al resultado calculado anteriormente se obtiene el valor del campo futuro $P^{n+1}_{i,j}$

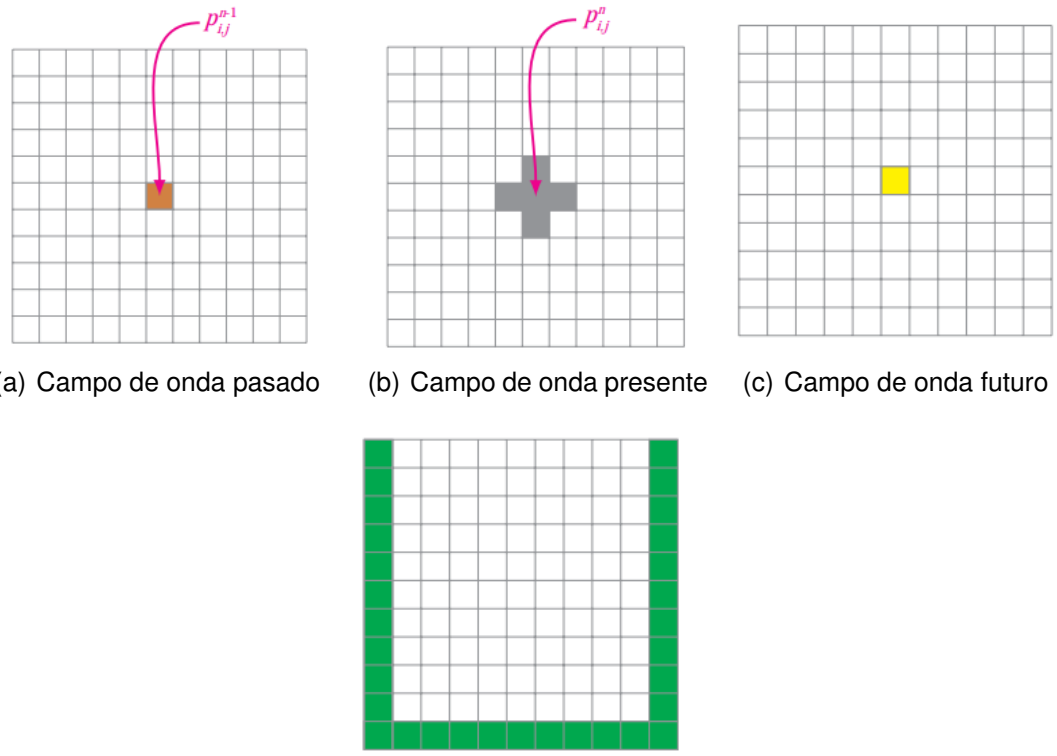
$$
\begin{aligned}
P_{temp} = \frac{1}{\Delta h^2}[-\frac{1}{560}\left(P_{i-4,j} + P_{i+4,j}\right) + \frac{8}{315}\left(P_{i-3,j} + P_{i+3,j}\right) - \frac{1}{5}\left(P_{i-2,j} + P_{i+2,j}\right) \\
+\frac{8}{5}\left(P_{i-1,j} + P_{i+1,j}\right) - \frac{205}{72}P_{i,j}], \\
R_{temp} = \frac{1}{\Delta h^2}[-\frac{1}{560}\left(R_{i,j-4} + R_{i,j+4}\right) + \frac{8}{315}\left(R_{i,j-3} + R_{i,j+3}\right) - \frac{1}{5}\left(R_{i,j-2} + R_{i,j+2}\right) \\
+\frac{8}{5}\left(R_{i,j-1} + R_{i,j+1}\right) - \frac{205}{72}R_{i,j}],
\end{aligned}
\tag{13}
$$

$$
P_{i,j}^{n+1} = 2P_{i,j}^{n} - P_{i,j}^{n-1} + \Delta t^2\left(c_x P_{temp} + c_z R_{temp}\right). \tag{14}
$$

Se quiere calcular el factor $P_{i,j}^{n+1}$ para poder avanzar temporalmente en la propagación. La figura 2 muestra una representación gráfica de la implementación de la ecuación 13.

Para calcular un punto del campo de onda propagado en el futuro $P_{i,j}^{n+1}$ 2(c), se requiere información de cinco puntos del campo de onda presente 2(b) y un punto en el campo de onda pasado 2(a). Dichos puntos se suponen conocidos.

Figura 2: Representación gráfica de la propagación implementada. Imagen tomada de [2]



(a) Campo de onda pasado (b) Campo de onda presente (c) Campo de onda futuro

(d) Zona CPML

La misma plantilla se homologa para la propagación del campo de onda R.

Así como se discretizó la ecuación de campo de onda 3, también se debe discretizar la ecuación de campo de onda que incluye las fronteras atenuantes 4.

La implementación de la ecuación 60 se hace de manera similar a como se implementó la ecuación 59. La discretización se da en el anexo 8 con la ecuación 60

$$P_{temp} = \frac{1}{\Delta h^2}[-\frac{1}{560}\left(P_{i-4,j} + P_{i+4,j}\right) + \frac{8}{315}\left(P_{i-3,j} + P_{i+3,j}\right) - \frac{1}{5}\left(P_{i-2,j} + P_{i+2,j}\right) + \frac{8}{5}\left(P_{i-1,j} + P_{i+1,j}\right) - \frac{205}{72}P_{i,j}],$$

$$R_{temp} = \frac{1}{\Delta h^2}[-\frac{1}{560}\left(R_{i,j-4} + R_{i,j+4}\right) + \frac{8}{315}\left(R_{i,j-3} + R_{i,j+3}\right) - \frac{1}{5}\left(R_{i,j-2} + R_{i,j+2}\right) + \frac{8}{5}\left(R_{i,j-1} + R_{i,j+1}\right) - \frac{205}{72}R_{i,j}], \tag{15}$$

$$\delta\psi_{temp} = \frac{1}{\Delta h}(\frac{1}{280}\left(\psi_{i,j-4} - \psi_{i,j+4}\right) + \frac{4}{105}\left(\psi_{i,j+3} - \psi_{i,j-3}\right) + \frac{1}{5}\left(\psi_{i,j-2} - \psi_{i,j+2}\right) + \frac{4}{5}\left(\psi_{i,j+1} - \psi_{i,j-1}\right)), \tag{16}$$

$$\begin{aligned} \zeta_{x_{temp}} &= a_x\zeta^{n-1} + b_x\left[(P_{temp})^n + (\delta\psi_{temp})^n\right], \\ \zeta_{z_{temp}} &= a_z\zeta^{n-1} + b_z\left[(R_{temp})^n + (\delta\psi_{temp})^n\right], \end{aligned} \tag{17}$$

$$P_{i,j}^{n+1} = 2P_{i,j}^{n} - P_{i,j}^{n-1} + \Delta t^2(c_x(P_{temp} + \delta\psi_{temp} + \zeta_{x_{temp}}) + c_z(R_{temp} + \delta\psi_{temp} + \zeta_{z_{temp}})). \tag{18}$$

Del mismo modo para implementar el campo de onda R. Para el cálculo de la variable temporal $\delta\psi_{temp}$ se debe desarrollar la ecuación dada para la variable auxiliar ψ en la ecuación 5 para las dos dimensiones(x y z).

2.3 INVERSIÓN DE ONDA COMPLETA

La inversión de onda completa ha tenido una transición durante los últimos 20 años, pasando de ser usada exclusivamente en la academia a ser implementada en la exploración sísmica y de hidrocarburos. Gracias a este interés en la técnica se han desarrollado estrategias que generan soluciones con mayor precisión. Con el desarrollo de arquitecturas en paralelo, la FWI se ha podido implementar de forma práctica debido a que los tiempos de procesamiento se ven reducidos. Así la FWI pudo superar a otras técnicas de determinación de modelos de velocidades, como por ejemplo, la tomografía basada en rayos en términos de la resolución de pequeñas

estructuras complejas en tierra [12], o la técnica de tomografía de refracción y reflexión, que utiliza apenas la cinemática del tiempo de tránsito de los datos sísmicos [14].

Recientemente modelos de tierra invertidos comercialmente para la exploración del subsuelo empezaron a incorporar la isotropía transversal vertical (VTI) [29]. Los modelos con este tipo de isotropía requieren que la ecuaciones de onda utilizadas para la propagación y retropropagación contengan parámetros de anisotropía como ϵ y δ.

La FWI es popular por su capacidad para estimar modelos del subsuelo generalmente con mayor resolución[19] que muchos otros métodos, como la tomografía de tiempo de viaje y análisis de la velocidad de migración (MVA) [25].

La FWI es un proceso en el que dado un modelo de parámetros de entrada (m), se busca un modelo preciso del subsuelo en términos de parámetros como la velocidad, densidad y anisotropía, emulando el diseño de adquisición en campo. Esta función $G(m)$, es un algoritmo de optimización local no lineal basado en la retropropagación de las diferencias entre los datos reales y los datos modelados a través del modelo en sí [5].

En la FWI se asume inicialmente que se puede resolver la ecuación

$$d = G(m), \tag{19}$$

con G como función que describe como calcular los datos sísmicos dado un modelo m y d contiene el campo de ondas sísmicos predicho en ciertas partes del modelo.

La FWI es un algoritmo que utiliza la aplicación repetida del problema directo dado en la ecuación 19 para resolver el problema inverso no lineal

$$G^{-1}(d) = m', \tag{20}$$

en donde ahora d contiene los datos observados en campo y m' es el modelo que se está tratando de encontrar.

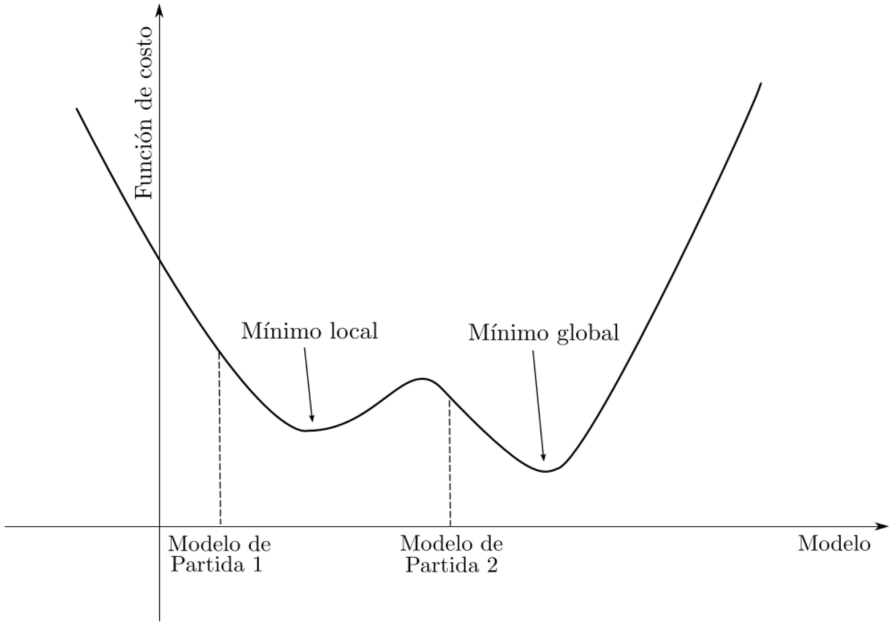
Para la implementación del algoritmo, G es una función en el dominio del tiempo que desarrolla la ecuación de onda con anisotropía VTI 2D expresada en diferencias finitas y que contiene también una definición del diseño de adquisición en campo e información sobre la fuente de ondas de campo.

Aunque se pueden desarrollar soluciones precisas y explicitas de la ecuación 19, no es posible describir soluciones a la ecuación 20. Esto, debido a que no es una función lineal. Por lo tanto, no tiene una forma inversa formal. En cambio, se puede usar una solución aproximada a la ecuación 20 y luego por medio de iteraciones buscar mejorar esta solución [31].

La FWI presenta problemas de unicidad, es decir que no tiene una única solución. El operador G^{-1} en la ecuación 20, al no ser lineal, presenta varios mínimos locales y un mínimo global como se ilustra en la figura 3. Es por esto que la convergencia de la FWI hacia el mínimo global, depende de los parámetros iniciales.

Para mitigar este problema, se han propuesto los enfoques multiescala [4]. Estos métodos multiescala añaden de forma recursiva los detalles de mayor frecuencia a los primeros modelos calculados a partir de datos de baja frecuencia.
La fidelidad de las técnicas multiescala depende fundamentalmente de la fidelidad del contenido de baja frecuencia en los datos registrados [11].

Figura 3: Mínimo local y mínimo global de una función de costo no lineal. Imagen tomada de [1]



El método FWI requiere las ecuaciones de onda para encontrar el campo de presión propagado hacia adelante y hacia atrás, una función de costo $\phi(m)$ y un método de actualización para estimar un modelo sísmico de velocidad. La función costo $\phi(m)$ usa la norma $l2$ al cuadrado de la diferencia entre el dato modelado y el observado para calcular el error de la estimación y se define como

$$\phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{d}(\mathbf{x} = x,\ = 0,\ t|m) - d^{obs}(\mathbf{x} = x,\ z = 0,\ t)\|_2^2 \tag{21}$$

donde $\mathbf{d}^{obs}(\mathbf{x} = x, z = 0, t)$ representa los datos observados en forma vectorizada en una localización espacial dada $(\mathbf{x} = x, z = 0)$, y $\mathbf{d}(\mathbf{x} = x, z = 0, t|m)$ representa los datos modelados en la mismas coordenadas espaciales, usando un modelo de velocidad conocido m. El dato modelado usado para calcular el error en la ecuación

21, puede ser obtenido usando la ecuación de onda 2D acústica con anisotropía VTI.

Se puede obtener una actualización del modelo de velocidad utilizando la ecuación

$$m^{k+1} = m^k - \alpha_k(H(m^k)^{-1}g(m^k)), \tag{22}$$

donde k denota la iteración del proceso de inversión, α_k es un escalar que determina el paso de avance hacia el mínimo de la función de costo. La inversa de la matriz Hessiana $H(m^k)^{-1}$ está relacionada con la información de curvatura de la función de costo y el gradiente $g(m^k)$ está relacionado con la dirección de actualización del modelo.

La inversa de la matriz Hessiana $H(m^k)^{-1}$ no suele ser calculada de manera exacta debido a su gran tamaño y alto costo computacional [13]. Por esta razón, se plantea un método general para el cálculo del siguiente modelo de velocidad

$$m^{k+1} = m^k - \alpha_k h(m^k), \tag{23}$$

donde $h(m^k)$ es la dirección de avance en la curva de la función de costo y se determina por medio de métodos como el gradiente descendiente dado en la sección 3.1 o el método quasi-Newton L-BFGS explicado en la sección 3.2.

El gradiente se puede calcular por medio del método del estado adjunto partiendo de la ecuación de propagación de onda 3.
Se requieren tres ingredientes para aplicar el método del estado adjunto: Las ecuaciones de estado, que son las mismas ecuaciones de propagación, las ecuaciones adjuntas, determinadas con las ecuaciones de propagación y el objetivo funcional, es decir la función de costo, la normal $l2$ al cuadrado del error [30].

De acuerdo a [32] las ecuaciones de onda 3 ya discretizadas en espacio, se pueden

escribir de manera simplificada como

$$T(m)\ddot{s} - C(m)\dot{s} - A(m)s - b(m) = 0, \tag{24}$$

donde T, C y A son matrices de coeficientes que se desprenden de la discretización en espacio y $b(m)$ contiene los términos de fuente. Los términos de s representan los campos de onda usados en el operador de onda y el punto sobre los campos representa la derivada en el dominio del tiempo.

Cada termino se puede definir como

$$T(m)\ddot{s} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{P} \\ \ddot{R} \end{bmatrix}, \tag{25}$$

$$C(m)\dot{s} = 0. \tag{26}$$

$$A(m)s = \begin{bmatrix} c_x \frac{\delta^2}{\delta x^2} & c_z \frac{\delta^2}{\delta z^2} \\ d_x \frac{\delta^2}{\delta x^2} & d_z \frac{\delta^2}{\delta z^2} \end{bmatrix} \begin{bmatrix} P \\ R \end{bmatrix}, \tag{27}$$

$$b(m) = f. \tag{28}$$

El termino $C(m)\dot{s}$ es cero pues no se tiene ninguna derivada de primer orden en las ecuaciones de propagación. donde f es la ondícula fuente.
La ecuación adjunta obtenida aplicando el lagrangiano a la derivada de la ecuación de costo, es dada por Wesdorp en [32] como

$$\ddot{\lambda}^T \frac{\delta h}{\delta \ddot{s}} + \dot{\lambda}^T \left(2\frac{\delta}{\delta t}\frac{\delta h}{\delta \ddot{s}} - \frac{\delta h}{\delta \dot{s}} \right) + \lambda^T \left(\frac{\delta h}{\delta s} + \frac{\delta^2}{\delta t^2}\frac{\delta h}{\delta \ddot{s}} - \frac{\delta}{\delta t}\frac{\delta h}{\delta \dot{s}} \right) = -\frac{\delta f}{\delta s}, \tag{29}$$

donde λ es el adjunto, $\frac{\delta h}{\delta \ddot{s}}$ es la derivada del sistema de ecuaciones con respecto a los campos de onda y $\frac{\delta f}{\delta s}$ es el residual. El residual se puede definir como la derivada de la fuente con respecto a los campos de onda. Se hace la aclaración que las derivadas en el dominio del tiempo son derivadas totales, mientras que las las derivadas respecto a los campos de onda son derivadas parciales. La ecuación 29 es llamada la ecuación adjunta del sistema de ecuaciones de onda h. La ecuación se puede aplicar al sistema de ecuaciones de onda para un medio VTI, ecuaciones tratadas en este trabajo, obteniendo

$$T^T(m)\ddot{\lambda} = -C^T(m)\dot{\lambda} + A^T(m)\lambda - \left(\frac{\delta f}{\delta s}\right)^T. \tag{30}$$

Dado $T(m)$ en la ecuación 25, la transpuesta de esta matriz es ella misma. Entonces se puede decir que $T^T(m) = T(m)$. El termino $C^T(m)\dot{\lambda}$ es nuevamente cero y la transpuesta de la matriz $A(m)$ está dada por

$$A^T(m)\lambda = \begin{bmatrix} \frac{\delta^2}{\delta x^2}c_x & \frac{\delta^2}{\delta z^2}d_x \\ \frac{\delta^2}{\delta x^2}c_z & \frac{\delta^2}{\delta z^2}d_z \end{bmatrix} \begin{bmatrix} \lambda_P \\ \lambda_R \end{bmatrix}. \tag{31}$$

De esta manera obtenemos la ecuación adjunta para el set de ecuaciones de onda con anisotropía VTI

$$\begin{aligned} \ddot{\lambda_P} &= \frac{\delta^2}{\delta x^2}(c_x\lambda_P) + \frac{\delta^2}{\delta z^2}(d_x\lambda_R) + Res, \\ \ddot{\lambda_R} &= \frac{\delta^2}{\delta x^2}(c_z\lambda_P) + \frac{\delta^2}{\delta z^2}(d_z\lambda_R) + Res. \end{aligned} \tag{32}$$

Aplicando el lagrangiano a la ecuación 24 se encuentra la siguiente expresión general para el gradiente total respecto a todos los parámetros del modelo

$$\frac{\delta\chi}{\delta m} = \int_0^T \lambda \left(\frac{\delta T}{\delta m}\ddot{s} - \frac{\delta C}{\delta m}\dot{s} - \frac{\delta A}{\delta m}s - \frac{\delta b}{\delta m} \right) \delta t, \tag{33}$$

donde $\frac{\delta\chi}{\delta m}$ es el gradiente. $\frac{\delta T}{\delta m}$, $\frac{\delta C}{\delta m}$ y $\frac{\delta b}{\delta m}$ son iguales a 0 pues no depende de los parámetros del modelo, por lo que queda

$$\frac{\delta\chi}{\delta m} = -\int_0^T \lambda \frac{\delta A}{\delta m} s dt. \tag{34}$$

Pero el gradiente total se puede escribir como la suma de los gradientes parciales como

$$\frac{\delta\chi}{\delta m} = \frac{\delta\chi}{\delta c_x} + \frac{\delta\chi}{\delta c_z} + \frac{\delta\chi}{\delta d_x} + \frac{\delta\chi}{\delta d_z}, \tag{35}$$

entonces tenemos que cada gradiente parcial respecto a los parámetros del modelo manejados para las ecuaciones VTI se puede expresar como

$$\frac{\delta\chi}{\delta c_x} = -\int_0^T \begin{bmatrix} \lambda_P & \lambda_R \end{bmatrix} \begin{bmatrix} \frac{\delta^2}{\delta x^2} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} P \\ R \end{bmatrix} dt, \tag{36}$$

ahora, obtenemos la expresión final para el gradiente respecto a c_x

$$\frac{\delta\chi}{\delta c_x} = -\int_0^T \lambda_P \frac{\delta^2 P}{\delta x^2} dt, \tag{37}$$

de la misma manera se hallan los gradientes con respecto a los dos parámetros faltantes c_z y $d_x = d_z$

$$\frac{\delta\chi}{\delta c_z} = -\int_0^T (\lambda_P + \lambda_R) \frac{\delta^2 R}{\delta z^2} dt, \tag{38}$$

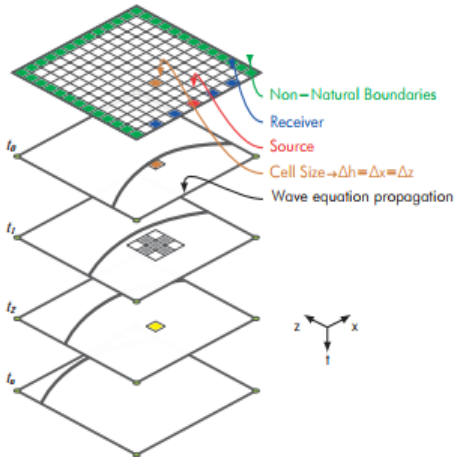
$$\frac{\delta\chi}{\delta d_x} = -\int_0^T \lambda_R \frac{\delta^2 P}{\delta x^2} dt, \tag{39}$$

en los cuales los campos de onda P y R son los volúmenes de datos obtenidos a partir de la propagación de fuentes puntuales en la superficie y el campo de onda λ es el volumen de datos obtenidos a partir de la ecuación adjunta, pero usando como fuente el residual como se puede ver en la ecuación 29. Esta propagación se realiza en el tiempo inverso convirtiendo a cada geófono en fuentes. Por esta razón, λ se llama también el campo retropropagado del residual.

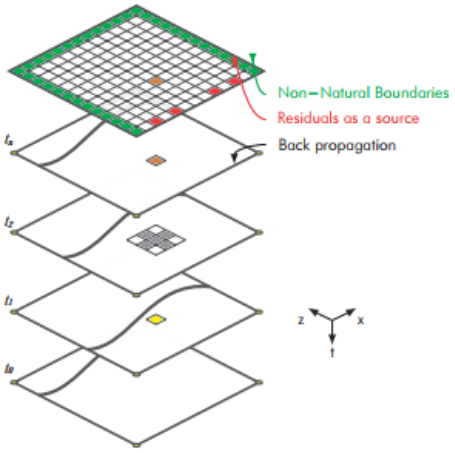
La FWI actualiza iterativamente los modelos de velocidades con la ecuación 23. El modelo de velocidades dado por la FWI puede ser utilizado para la caracterización de reservatorios o para mejorar el resultado de la Migración de tiempo inverso (RTM). La FWI puede ser incluida en un flujo de trabajo para la generación de imágenes en profundidad proveyendo mejores modelos de entrada para la RTM, en consecuencia, mejorando los resultados del proceso de generación de imágenes.

La figura 4(a) muestra el campo de propagación generado por una fuente, donde los geófonos reciben la información del campo de presión. El error entre los datos registrados por los geófonos, mejor conocidos como trazas sísmicas y los datos modelados, se utiliza como fuente para hallar el campo retropropagado del residual por medio de la retropropagación también mostrada en la figura 4(b).

Figura 4: Propagación y retropropagación del campo de ondas. Imagen tomada de [2]



(a) Propagación de una fuente (en color rojo) a diferentes pasos de tiempo utilizando la ecuación de onda. Los geófonos se muestran en azul y la zona límite o zona CPML se representan en verde.



(b) Retropropagación de los residuales (en rojo) en diferentes pasos de tiempo utilizando la ecuación de onda. La zona límite se representa en verde.

3. DESCRIPCIÓN DEL ALGORITMO DE INVERSIÓN DE ONDA COMPLETA

El algoritmo se divide en 5 pasos para cada iteración siendo estos:

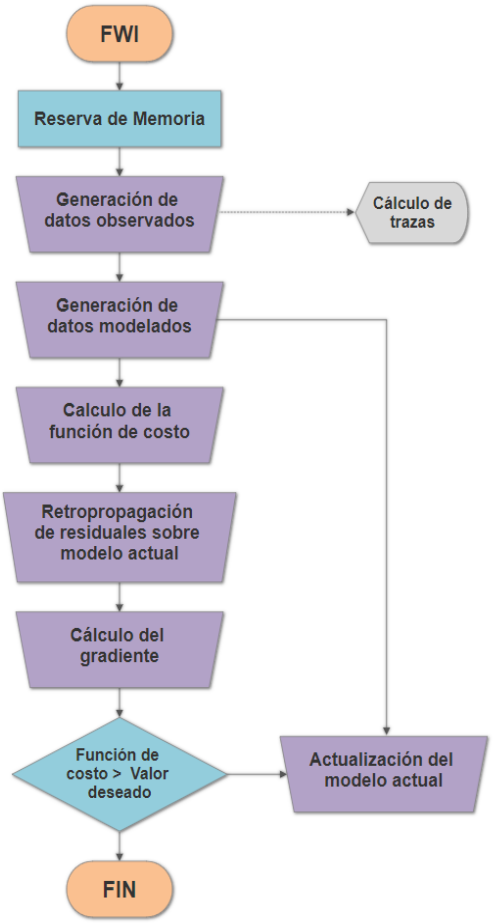
1. Generación del dato observado.
2. Generación del dato modelado por medio de propagación sobre el modelo de velocidad inicial.
3. Retropropagación de los residuales sobre el modelo inicial.
4. Cálculo del gradiente.
5. Actualización del modelo inicial con el modelo resultante del gradiente.

Estas operaciones se repiten una cierta cantidad de veces hasta el punto donde la función de costo o error llega a un valor deseado.

De la figura 5 se pueden hacer varios análisis en cuanto a la ejecución del algoritmo. Cada uno de los bloques trapezoidales representa un punto crítico de ejecución y realiza operaciones matemáticas entre matrices. Para ciertos pasos de la FWI como la propagación del campo de onda, la cantidad de operaciones por paso es $nx * nz * nt$. Esto constituye un gran volumen de operaciones para tiempos extensos de propagación y modelos matricialmente grandes.

Los parámetros de entrada de la FWI son el dato observado d, el modelo obtenido en campo m y la ondícula fuente f.
El primer paso de ejecución es la propagación que consiste en la solución de la ecuación 3 con fuentes puntuales $f(nt)$ para cada uno de los disparos nt. La solución genera el campo de onda $p_s(x, z, t)$, usado para obtener las trazas modeladas, que son la primera fila del campo de onda p_s en cada capa de tiempo.

Figura 5: Flujograma de la FWI con las etapas del algoritmo implementado.



El siguiente paso es el cálculo de los residuales d_{res}, que se calculan restando las trazas obtenidas anteriormente para el modelo inicial y modelo obtenido en campo.

$$d_{res} = d_{mod}(m) - d_{obs}(m) \tag{40}$$

Los residuales se usaran como fuente para la retropropagación. La retropropagación se hace a partir de las ecuaciones del adjunto 32. Se propaga en tiempo inverso (desde la ultima muestra registrada hacia la primera) y se obtiene el campo retropropagado λ. Con el campo propagado y el campo retropropagado se realiza

el cálculo del gradiente con la ecuación 34.

Una vez calculado el gradiente se hace la actualización del modelo con la ecuación 23. Se debe emplear alguna técnica para determinar la dirección de avance $h(m^k)$.

En el algoritmo se implementa un enfoque multiescala con frecuencias centrales de 3, 6 y 9 Hz, donde el modelo final obtenido por cada frecuencia anterior, se usa como punto de partida para la siguiente frecuencia, con el fin de resolver primero los componentes frecuenciales bajos, asociados a las formas del modelo y luego los componentes frecuenciales altos, asociados a los detalles presentes en el modelo.

3.1 MÉTODO DEL DESCENSO DEL GRADIENTE

Una vez que el gradiente de la función de costo es obtenido utilizando el método del estado adjunto, la forma más sencilla de obtener una dirección de búsqueda es a partir del gradiente utilizando el método del gradiente descendente, en donde la dirección de búsqueda $h(m^k)$ es el gradiente de la función de costo, es decir

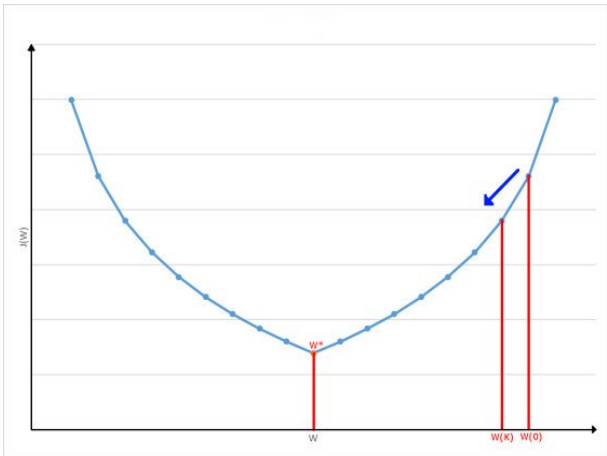
$$h(m^k) = g(m^k). \tag{41}$$

El método utiliza el gradiente de una función evaluado en un punto para indicar la dirección en que la función disminuye más rápidamente desde el punto de interés [15].
Reemplazando la ecuación 41 en la ecuación 23 se obtiene la nueva función para actualizar el modelo de velocidades

$$m^{k+1} = m^k - \alpha_k g(m^k). \tag{42}$$

El método de actualización se basa en el descenso del gradiente para las primeras iteraciones del algoritmo. Para cada iteración al modelo inicial se le resta el gradiente escalado α veces. La selección de este valor α define el paso del descenso en la

Figura 6: Representación gráfica del método del descenso del gradiente.



curva de desajuste no lineal (véase figura 3). Para valores grandes la reducción del error se hace mas rápida. La desventaja de usar valores muy grandes para α, es que una vez el error se ve reducido y estamos situados en una posición cercana a un mínimo, es posible que los valores del gradiente oscilen entre algunos puntos al rededor del mínimo.

Dependiendo del modelo inicial o modelo obtenido, se puede llegar a un mínimo local o global de la función de costo. Por tanto es recomendable tener un buen criterio de selección para el modelo inicial.

La figura 6 es una buena representación del método del descenso del gradiente. La flecha azul representa la dirección en la que se mueve el método por la curva. Luego cada una de las divisiones de la curva es representada con un punto azul, la distancia que separa a cada uno de estos puntos es α. El punto w, es el mínimo al que se desea llegar. En cada iteración el modelo avanza a lo largo de la curva, este se actualiza y el proceso se repite hasta llegar al mínimo.

3.2 MÉTODO DE L-BFGS

Anteriormente se manejaba un α contante para hacer la actualización del modelo, el α constante define un paso constante hacia el mínimo en la función de costo.

Para hacer el proceso de inversión más eficaz, se implementa un método cuasi-Newton, el cual aporta información respecto a la dirección de búsqueda y a su vez también aporta información sobre la curvatura de la función de costo asociada a los datos modelados sobre m^k.

El método de BFGS, llamado así por sus desarrolladores Broyden, Fletcher, Goldfarb y Shanno, propone guardar todos los modelos y gradientes obtenidos en cada una de las iteraciones realizadas en la inversión, y luego operarlos para obtener una dirección de búsqueda $h(m^k)$ que reemplaza al gradiente en la ecuación 42 obteniendo así

$$m^{k+1} = m^k - \alpha_k h(m^k). \tag{43}$$

Guardar los modelos y los gradientes para aplicar el método BFGS representa un aumento en el costo computacional de la FWI. Para disminuir este costo, se implementa el método L-BFGS, que limita el número de gradientes y modelos de velocidad que se guardan para el cálculo de $h(m^k)$, ya que las iteraciones iniciales dejan de dar información importante para el proceso de minimización.

El método L-BFGS utiliza los gradientes y modelos de las l iteraciones anteriores (típicamente con $l < 20$) para calcular una aproximación simple y compacta del producto de la inversa de la Hessiana con el gradiente $H(m^k)^{-1}g(m^k)$ [15], tal como se presenta en el Algoritmo 1, algoritmo tomado de [1].

Algoritmo 1 L-BFGS Method

$q \leftarrow g(m^k)$; ▷ Iteración k de la inversión
for $i = k-1 : -1 : k-m$ **do**
    $\epsilon_i \leftarrow \sigma_i s_i^T q$; ▷ s_i diferencia de modelos de velocidad en el punto$i$.
    $q \leftarrow q - \epsilon_i y_i$; ▷ y_i es un operador de escala en el punto i.
end for
$r \leftarrow D_k^0 q$;
for $i = k-m : 1 : k-1$ **do**
    $\beta \leftarrow \sigma_i y_i^T r$;
    $r \leftarrow r + s_i(\epsilon_i - \beta)$;
end for
return ϕ, m_{final} ▷ Resultados de la FWI

En Algoritmo 1, s_k representa la diferencia entre los modelos de velocidad

$$s_k = m^{k+1} - m^k, \tag{44}$$

y_k sirve como un operador de escala, que trata de emular valores verdaderos de la matriz Hessiana y se define como

$$y_k = g(m^{k+1}) - g(m^k); \tag{45}$$

Emulando el producto de la Hessiana con el gradiente, el método L-BFGS calcula mejores y mas precisas direcciones de descenso que el método del gradiente descendente [24].
El método de L-BFGS, se implementa con el algoritmo 1 y el conjunto de ecuaciones dadas a continuación

$$\sigma_k = \frac{1}{\langle s_k, y_k \rangle}, \tag{46}$$

$$D_k^0 = \gamma_k I, \tag{47}$$

$$\gamma_k = \frac{\langle s_k, y_k \rangle}{\langle y_k, y_k \rangle}. \tag{48}$$

Este proceso se implementa en cada iteración y la nueva dirección de búsqueda del siguiente modelo de velocidad esta determinada por r_k. La ecuación que define el siguiente modelo de velocidades es

$$m^{k+1} = m^k - \alpha_k r_k, \tag{49}$$

donde el valor de α_k es escogido tal que cumpla la condición

$$\phi(m^k - \alpha_k r_k) < \phi(m^k). \tag{50}$$

Se empieza normalmente con un $\alpha_k = 1$, y disminuyendo este valor a la mitad si no cumple la condición definida en la ecuación 50 hasta que sea cumplida, recordando que $\alpha_k \in (0, 1]$.

El requisito principal para la implementación del método es contar con un histórico de al menos dos modelos de velocidad y dos gradientes calculados en iteraciones anteriores. Por esta razón, las primeras iteraciones de la FWI se realizan usando el método del gradiente descendente. Se debe cumplir que el modelo obtenido en estas primeras iteraciones, debe tener un error menor en comparación al error obtenido con la iteración que usó el modelo de partida, es decir se debe tener una disminución del error desde la primera iteración hasta la iteración donde se aplica L-BFGS.
Con los modelo obtenidos en las iteraciones anteriores y el gradiente de cada uno de estos modelos, se puede realizar la implementación del L-BFGS para las siguientes iteraciones de la inversión.

4. PROBLEMAS EN LA IMPLEMENTACIÓN COMPUTACIONAL DE LA FWI

Dentro de esta sección se plantean los principales problemas de la implementación de la FWI a nivel computacional. Se hace referencia a las exigencias del algoritmo y las limitantes del hardware sobre el cual se va a implementar. Posteriormente se hace un análisis breve sobre arquitecturas paralelas y secuenciales presentando las ventajas de trabajar de forma paralela.

Hay dos principales problemas en la implementación computacional del método de inversión de onda completa. La naturaleza del algoritmo y las ecuaciones utilizadas por el mismo demandan un volumen muy grande de operaciones, véase las ecuaciones de la sección 2.2. Estas ecuaciones presentan derivadas espaciales y temporales, lo cual se traduce a múltiples operaciones matriciales(véase figura 4). El volumen de datos y operaciones en este caso depende del tamaño del modelo y tiempo de propagación. Por tanto, los principales problemas para implementar este algoritmo son, el volumen de datos y operaciones necesarias para propagar y retropropagar.

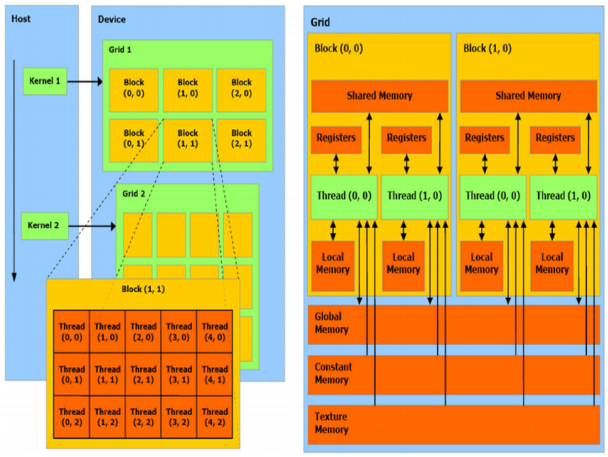
El algoritmo se implementa usando una arquitectura en paralelo debido a la cantidad de operaciones para cada iteración de la FWI. Puesto que, las arquitecturas en paralelo permiten hacer operaciones de forma simultanea, se reducen considerablemente los tiempos de ejecución comparado con arquitecturas secuenciales.

4.1 ARQUITECTURA DE LA GPU

Nvidia, uno de los principales fabricantes de GPU's, maneja una arquitectura de múltiples unidades aritméticas lógicas (Alu por sus siglas en ingles *Arithmetic logic unit*) para ejecutar operaciones de forma simultanea. A esta arquitectura se le denomina arquitectura de calculo paralelo. La cantidad de operaciones que se pueden hacer simultáneamente dependen directamente del modelo de GPU.
Para este proyecto el modelo utilizado es la tarjeta Nvidia QUADRO PNY K4200 y 2

placas aceleradoras NVIDIA TESLA K40 AC. Estas dos GPU, mantienen la misma arquitectura con diferencias en los disipadores.

Figura 7: Representación gráfica de la arquitectura de una GPU[16].



En la figura 7 podemos ver una representación gráfica de la arquitectura seleccionada para la implementación de este algoritmo. Se trabajará con un conjunto de herramientas proporcionadas por Nvidia llamada CUDA. Nvidia facilita un grupo de herramientas para desarrolladores, entre los que se encuentra un compilador que secciona el código en dos partes, una parte que se ejecuta en la GPU(*Device*) y otra parte que se ejecuta en la CPU(*Host*).

4.2 VOLUMEN DE OPERACIONES Y DATOS

La FWI es un algoritmo que debido a las ecuaciones que utiliza exige una gran cantidad de operaciones en el plano discreto. Esta cantidad de operaciones depende de dos factores importantes, el tamaño del modelo y el tiempo de propagación. Sabiendo esto podemos definir la cantidad de operaciones del modelo como

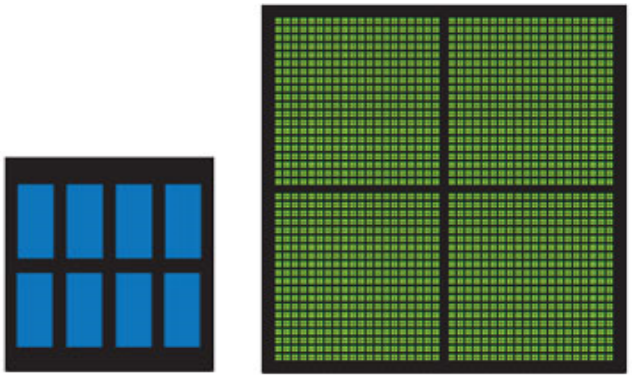
$$nx * nz * nt, \tag{51}$$

donde nx es la cantidad de muestras de nuestro modelo en el eje x y nz es la cantidad de muestras en z. Estas operaciones se repiten de forma iterativa nt veces, donde nt depende de la frecuencia de muestreo y del tiempo de propagación.

Puesto que la cantidad de operaciones crece dependiendo del tiempo de propagación en un factor de $nx * nz$ se hace muy complicado ejecutar este algoritmo de forma rápida en arquitecturas secuenciales como la CPU.

Una forma sencilla de entender la diferencia entre la GPU y la CPU es comparar la forma en que procesan las tareas. Una CPU está formada por varios núcleos optimizados para el procesamiento en serie, es decir para realizar una tarea tras otra, esto implica que si una tarea se retrasa, el algoritmo completo se retrasa. Es aquí donde el volumen de operaciones se convierte en un problema. Una GPU consta de millares de núcleos más pequeños y eficientes diseñados para manejar múltiples tareas simultáneamente. Esto se puede ver en la figura 8.

Figura 8: Comparación de cantidad de núcleos entre la CPU y la GPU. Figura tomada de [17]



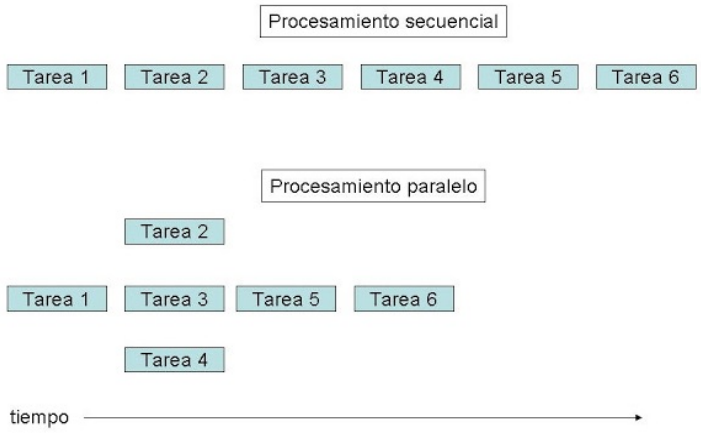
(a) Multiples núcleos de una CPU

(b) Miles de núcleos de una GPU

La FWI tiene partes que se pueden trabajar paralelamente y partes secuenciales. Una tarea se puede desarrollar de manera paralela si no tiene dependencia de datos es decir, los datos resultantes no serán necesarios para el calculo de datos futuros. Los $nx * nz$ cálculos realizados en cada iteración para la propagación y retropropagación se pueden desarrollar de forma paralela, solucionando así el problema del volumen de datos. Los pasos dados en la figura 5 se deben realizar de forma secuencial pues es necesario tener la totalidad de datos de la iteración presente para

el desarrollo de la iteración futura.

Figura 9: Desarrollo de tareas de manera secuencial y paralela.



4.2.1 Volumen de datos A medida que se desarrolla el algoritmo de inversión de onda se va generando un volumen de datos necesarios para diferentes partes del código, por ejemplo las trazas halladas con la propagación, son utilizadas para el cálculo del gradiente.

Cada iteración del algoritmo genera una matriz de $nx*nz$ datos con valores flotantes de 32 bits. Por lo tanto se tiene un total de $nx * nz * 4$ bytes por dato calculado. Se debe tener en cuenta que se genera la propagación y retropropagación, volúmenes que se deben almacenar para cálculos posteriores.

Las arquitecturas de computo paralelo como las GPUs tienen poca memoria para almacenamiento de datos. Además se debe tener en cuenta que los volúmenes de la propagación y retropropagación se deben almacenar para cálculos posteriores.

La GPU usada para implementar el algoritmo cuenta con 12GB DDR5, lo que limita el espacio de almacenamiento de los volúmenes generados y se convierte en punto de partida para la generación de diferentes estrategias de computo que den solución al problema de la memoria.

5. ESTRATEGIAS COMPUTACIONALES PARA LA IMPLEMENTACIÓN DE LA FWI

En el capítulo 4 se aclararon los principales problemas para implementar la FWI de forma computacional, puesto que se trabajará con una arquitectura en paralelo (GPU). El tiempo de ejecución se ve reducido, solventando el problema de periodos largos de ejecución debido al volumen de operaciones. Una de las principales limitantes de la GPU es la poca RAM que esta posee, por lo tanto modelos muy grandes de datos deben ser procesados aplicando algunas estrategias computacionales para que resulte posible operarlos con la GPU.

Se propusieron dos estrategias computacionales con el fin de ahorrar RAM en la GPU. Los dos métodos propuestos son “punto de referencia” o *checkpointing method* por su nombre en inglés y “latencia oculta”. El primer método secciona los pasos del algoritmo, por lo que la propagación se hace por partes. El segundo método aprovecha las transferencias asíncronas de datos con el fin de establecer una transferencia óptima entre el *host* y *device* y así aprovechar la memoria del host.

5.1 MÉTODO DEL PUNTO DE REFERENCIA

El gradiente que se usa para actualizar el modelo requiere datos de los campos de onda propagados y de un residual basado en las diferencias entre el campo de onda simulado y los datos obtenidos en campo [3]. Es por esto que para el desarrollo del algoritmo se necesita tener acceso a la información obtenida en cada paso de la FWI, ya sea la propagación o la retropropagación.
El método del punto de referencia, es una estrategia computacional diseñada para dar una solución al problema que se presenta por la falta de memoria para almacenar datos. Esta estrategia reduce la cantidad de memoria que se debe reservar en la GPU.

Inicialmente, en la implementación de la FWI se propaga sobre el modelo sintético con el fin de obtener los *shot gather* para cada fuente, también se hace la propaga-

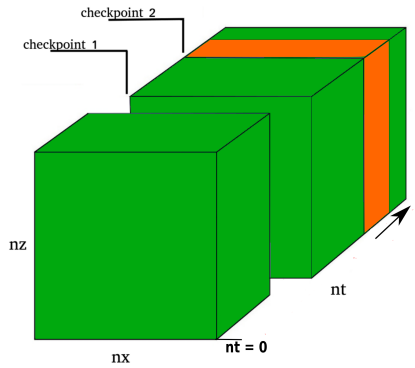
ción sobre el modelo obtenido en campo con el mismo fin. Posteriormente se halla un residual, siendo este el resultado de restar punto a punto los *shot gather* del modelo sintético y del modelo obtenido en campo. Durante este proceso es necesario almacenar todos los *frames* de la propagación del campo de onda. Es en este punto donde se aplica la estrategia para reducir la cantidad de datos almacenados durante la ejecución del algoritmo.

El cálculo de la FWI requiere la propagación sobre el modelo inicial y el modelo obtenido en campo para hallar el residual. Una vez se tiene el residual, se retropropaga para calcular el gradiente.
Para realizar el método del *checkpointing*, se hace una segmentación de datos y cálculos durante la propagación y retropropagación, es decir se deben hacer propagaciones y retropropagaciones parciales. En la estrategia, se hace la propagación sobre el modelo inicial y durante este proceso, se guardan los *checkpoints* que segmentarán los volúmenes de datos. Los *checkpoints* np dividen el volumen total de *frames* que se tienen en la propagación y retropropagación nt. Es por esto que se le llama segmentación. Cada volumen parcial de datos obtenido por la división del volumen total de datos entre el número de *checkpoints* se llama segmento. Para mejor comprensión llamaremos a estos segmentos s.

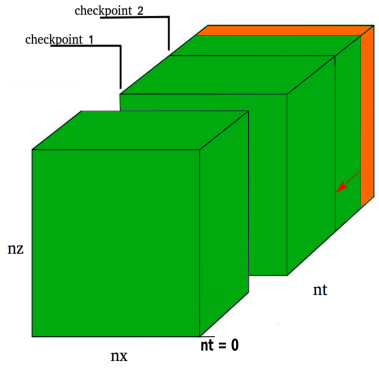
En la figura 10(a) se muestra que la propagación se hace a partir del ultimo segmento ns en el sentido que indica la flecha, para la siguiente iteración la propagación se hace en el segmento anterior $(n-1)s$ como se muestra en la figura 10(c).
Para la retropropagación se muestra en la figura 10(b) que también se empieza a propagar en el último segmento pero en sentido contrario a la propagación, es decir siguiendo la flecha roja. Se propaga desde el último *frame* hasta el primer *frame* de cada segmento. Es por esto que se llama retropropagación, pues va en sentido contrario al tiempo. En la siguiente iteración, se hace la retropropagación para el segmento anterior $(n-1)s$ como se muestra en la figura 10(d). Esto se repite hasta completar la propagación y retropropagación de todos los segmento, del volumen total de datos.

Esta estrategia, en una iteración, hace una propagación y retropropagación parcial, de nt/np puntos en cada segmento. Una vez calculado un segmento de propagación y retropropagación, se hace el cálculo del gradiente asociado.

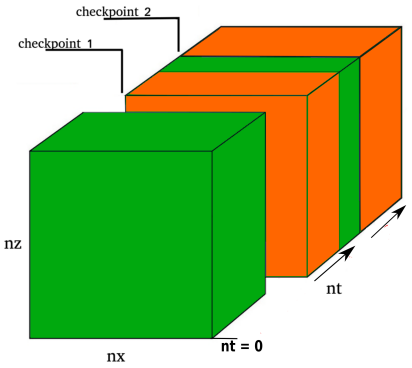
Figura 10: Sentido de propagación y retropropagación con *checkpoints*.



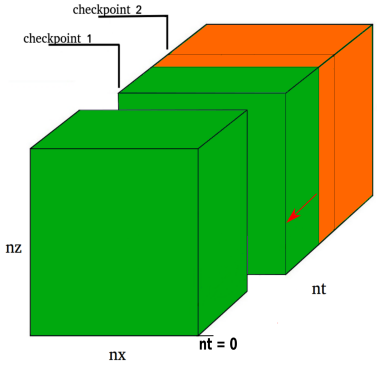
(a) Sentido de propagación a través del tiempo indicado por la flecha para el último segmento.



(b) Sentido de retropropagación a través del tiempo indicado por la flecha roja para el último segmento



(c) Sentido de propagación a través del tiempo indicado por la flecha para el segmento anterior$(n,1)S$.



(d) Sentido de retropropagación a través del tiempo indicado por la flecha roja desde nt_{final} hasta $nt = 0$ para el segmento $(n-1)s$.

Esta estrategia consiste en seleccionar una cantidad de puntos de guardado durante

la propagación del modelo inicial. La cantidad de memoria a reservar pasa de $nx * nz * nt * 4$ a ser

$$M_1 = \frac{nx * nz * nt * 4}{np}, \tag{52}$$

donde np es el número de puntos utilizados para hacer el *checkpointing* y M_1 la memoria a reservar. La selección del número de puntos de referencia o *checkpoints*, se debe hacer con cuidado, ya que seleccionar un número muy grande no resulta beneficioso puesto que para almacenar estos *checkpoints* es necesario reservar memoria. La cantidad de memoria necesaria para los *checkpoints* sería

$$M_2 = nx * nz * np * 4, \tag{53}$$

siendo M_2 la memoria necesaria para los *checkpoints* también llamada estado. Sabiendo esto el criterio de selección del número de puntos de referencia se hace muy importante para así poder reservar la menor cantidad de memoria que permita esta estrategia.

Teniendo las dos ecuaciones que involucran reserva de memoria (52 y 53), se puede encontrar la cantidad optima de *checkpoints* que mantenga un balance entre memoria reservada para los estados y memoria ocupada por los datos, hallando el mínimo de la función resultante de la suma de las ecuaciones (Ecuación 54).

$$M_1 + M_2 = 4 * nx * nz \left(\frac{nt}{np} + np \right); \tag{54}$$

Se hace la derivada de la ecuación 54 y se iguala a cero para encontrar el mínimo de la función y así establecer un número de *checkpoints* como

$$\frac{\delta}{\delta np}\left(\frac{M_1 + M_2}{4 * nx * nz}\right) = \frac{\delta}{\delta np}\left(\frac{nt + np^2}{np^2}\right), \tag{55}$$

$$\frac{np^2 - nt}{np^2} = 0, \tag{56}$$

$$np = \sqrt{nt}. \tag{57}$$

Lo que quiere decir que el mínimo de *checkpoints* depende exclusivamente de nt y se tiene el valor que permite segmentar el código sin ocupar demasiada memoria por los estados ni por los datos.

El algoritmo implementado para el desarrollo del método se ve resumido en Algoritmo 2.

Algoritmo 2 Checkpointing method

$np = \sqrt{nt}$ ▷ Se define el número de *checkpoints* (np).
$MemSpace = nt/np$ ▷ Variable para definir cantidad de *frames* por segmento.
for $i = 0, \cdots, i = nShots$ **do** ▷ nShots, número de shots para la propagación.
    cudaMemset(); ▷ Establecer memoria en ceros para propagación.
    for $it = 0, \cdots, i < nt$ **do**
        Sinthetic model propagation. ▷ Propagación del modelo sintético.
        Shot generation ▷ Se calculan los shots para cálculo de residual.
    end for
end for
for $k = 0, \cdots, k < nIter$ **do** ▷ nIter, Número de iteraciones para actualizar el modelo.
    cudaMemset(); ▷ Establecer ceros en memoria para los gradientes
    for $i = 0, \cdots, i <= nShots$ **do**
        cudaMemset(); ▷ Memoria en ceros para propagación de modelo real.
        for $it = 0, \cdots, it < nt$ **do**
            Field model propagation. ▷ Propagación del modelo adquirido en campo.
        end for
        Launching Residual Kernel ▷ Se halla el residual.
        $\phi = 0;$ ▷ Variable para la función de costo.
        for $j = 0, \cdots, j < nx * nt$ **do**
            $|residual|_2^2$ ▷ Normal l2 del residual para cálculo de ϕ.
        end for
        for $x = 0, \cdots, x < np$ **do**
            cudaMemcpy(); ▷ Copia de campos de Onda P para propagación segmentada.
            for $y = 0, \cdots, y < memSpace$ **do**
                Segmentated field model propagation ▷ propagación con *checkpoints*.
            end for
            for $y = 0, \cdots, y < memSpace$ **do**
                Segmentated backwards propagation ▷ Retropropagación con *checkpoints*.
                Gradient calculation ▷ Se calcula y almacena el gradiente.
            end for
        end for
    end for
    for $i = 0, \cdots, i =$ **do**
        Gradient normalization ▷ Normalizar el gradiente.
    end for
    Model Update ▷ Actualización del modelo.
end for

5.2 MÉTODO DE LATENCIA OCULTA

Esta estrategia consiste básicamente en hacer copia de información del *device* al *host* de manera asíncrona para así dar una solución al problema asociado a la memoria de las GPU. Para clarificar el concepto de copiado asíncrono, se parte de la idea del desarrollo de tareas de manera secuencial, lanzando un *kernel* que genera un volumen de datos con el campo de onda y luego copiando dicha información al *host*. Sin embargo realizarlo de esta manera haría que los tiempos de ejecución se elevasen. Es por esto que haciendo uso de funciones y herramientas del *toolkit* de cuda se puede proponer un método diferente, en el cual mientras un *kernel* hace su trabajo, se realicen las copias de información.

El método *Hidden Latency*, nombre en inglés de la estrategia, se desarrolla a partir del uso de *streams*. Un *stream* es una secuencia de operaciones que se ejecuta en un orden determinado[23]. El *stream* le da la capacidad a un flujo de instrucciones de ejecutarse de forma paralela. El uso de *streams* requiere que el código se desarrolle de forma concurrente, que genera la posibilidad de realizar múltiples operaciones simultáneamente. Esta concurrencia va más allá del paralelismo granulado de un *kernel* en Cuda.

Al declararse un *stream* se crea un flujo de ejecución en diferente contexto al flujo secuencial por defecto de un programa. Al declarar múltiples *streams* se crean múltiples contextos de ejecución que se lanzan en paralelo siempre y cuando se especifique dentro del código. El flujo por defecto del programa está asociado al *stream* 0 y este se bloquea hasta que los otros *streams* acaben su ejecución. De esta manera se tiene un mejor control de la concurrencia del código.

La concurrencia dentro de la FWI se propone con el fin de paralelizar las transferencias de datos y los *kernels* que procesan la propagación. Dentro del algoritmo se declaran y crean dos *stream* diferentes al *stream* por defecto. De esta manera todos los *kernels* que se ejecutan antes de las derivadas temporales se ejecutan dentro de un contexto paralelo a la transferencia de datos. Esto permite mantener en funcionamiento la GPU mientras se transfieren los datos de las derivadas temporales

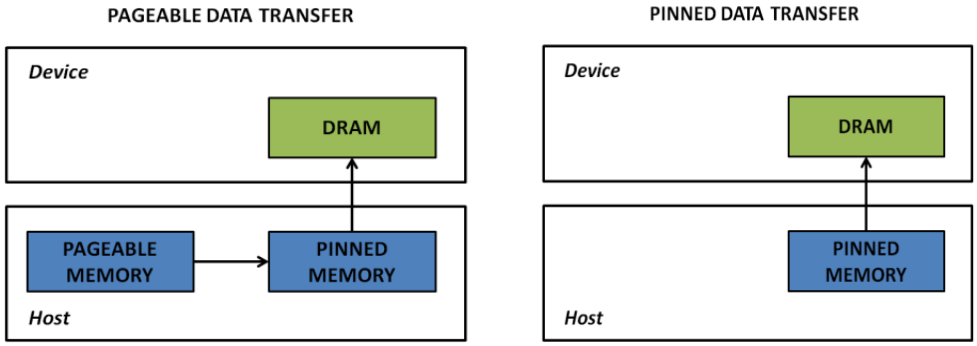
para posteriormente hacer el cálculo el gradiente.

Para asegurar la integridad de los datos a transferir se deben definir algunas condiciones y esto se logra identificando las dependencias entre los datos y los *kernel*. Por lo tanto, las transferencias de datos no se pueden ejecutar si los *kernel* que operan estos datos están en funcionamiento. Puesto que solo el *kernel* de derivadas temporales opera los datos relacionados a las transferencias, dicho *kernel* se ejecuta dentro del *stream* 0. De esta manera se asegura que este kernel no será ejecutado hasta que la transferencia de datos finalice.

A su vez se tiene requerimientos para asegurar la condición de concurrencia; dichos requerimientos exigen que las operaciones realizadas estén en diferentes *streams* y que estos *streams* no sean del tipo *stream* 0. Además, la memoria utilizada en el *host* para la transferencia de los datos debe ser tipo *pinned memory* y el *device* debe permitir hacer copias asíncronas bidireccionales.

La transferencia de datos entre *host* y *device* se hace normalmente, con el sistema de memoria virtual o en inglés *virtual memory system* (no *pinned memory*) en donde la información se copia a la *RAM* de manera sincronizada pero sin mantener un orden en la memoria física. Para copiar esta información el sistema accede a las diferentes direcciones de memoria, lo que puede llevar mucho tiempo. Usando *pinned memory*, la información es almacenada en la memoria física de la *RAM* de manera ordenada, reservando un espacio en memoria dedicado, lo que permite que se haga un acceso directo a memoria o DMA por sus siglas en inglés *Direct Memory Access* por parte del *device*, haciendo el proceso de copiado más rápido.

Figura 11: Comparación de transferencia de datos. Imagen tomada de [10]



(a) Transferencia de datos usando *virtual memory*

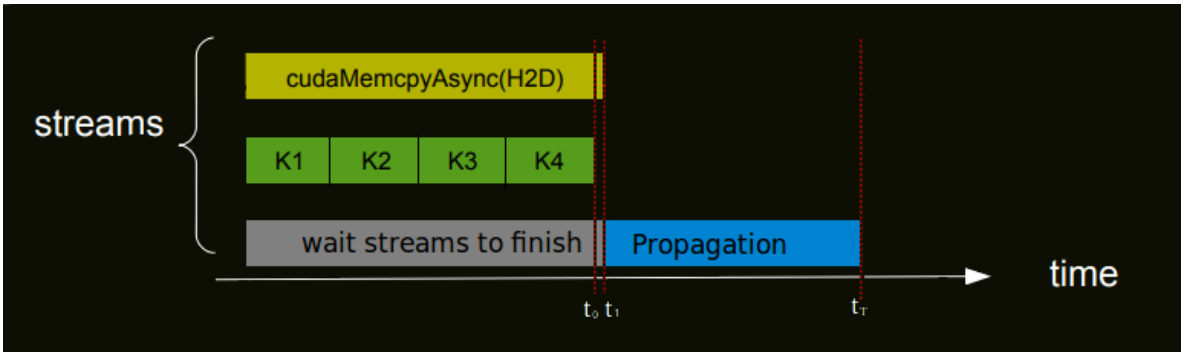
(b) Transferencia de datos con *pinned memory*

En el algoritmo se usa un *stream* para el lanzamiento de los *kernel* encargados de derivar el campo de onda P, derivada dada en la ecuación 3, y las variables auxiliares para el cálculo de la ecuación de onda que incluye la zona CPML dadas en 5.

Se utiliza otro *stream* para copiar el dato generado en cada paso de la propagación y retropropagación al *host*. Un ultimo *stream* es declarado en el algoritmo, el *stream* 0 que lanza el *kernel* bloqueante. En cada iteración se genera un nuevo dato de propagación y retropropagación. En la siguiente iteración, el *stream* encargado de la copia de datos, toma este dato generado en la iteración anterior y lo copia a la memoria del *host*. El proceso de lanzamiento de *kernels* y copiado de información se hace tanto para la propagación como para la retropropagación. Por lo tanto, cada uno de estos procesos tiene un *kernel* bloqueante.
Para la parte de propagación el *kernel* bloqueante es el encargado de la propagación y para la parte de la retropropagación el *kernel* bloqueante es el *kernel* que desarrolla el gradiente. Los *streams* trabajan de forma paralela cada uno con una duración diferente t_0 y t_1 como se ve en la imagen 12. El *kernel* lanzado por el *stream*0 es el *kernel* bloqueante, permite que los dos primeros *streams* terminen sus tareas para empezar la propagación y así asegurar que los datos no se corrom-

pan, es decir espera al tiempo t_1 antes de iniciar su ejecución.

Figura 12: *Streams* en desarrollo paralelo y *Kernel* bloqueante.



La implementación del algoritmo se puede ver resumida en Algoritmo3.

Algoritmo 3 Hidden Latency method

for $i = 0, \cdots, i = nShots$ **do** ▷ nShots, número de *shots* para la propagación.
  cudaMemset(); ▷ Establecer memoria en ceros para propagación.
  for $it = 0, \cdots, it < nt$ **do**
    Sinthetic model propagation. ▷ Propagación del modelo sintético.
    Shot generation ▷ Se calculan los *shots* para cálculo de residual.
  end for
end for
cudaStreamCreate(); ▷ Se crean los *streams*.
for $k = 0, \cdots, k < nIter$ **do** ▷ nIter, Número de iteraciones para actualizar el modelo.
  cudaMemset(); ▷ Establecer ceros en memoria para los gradientes
  for $i = 0, \cdots, i <= nShots$ **do**
    cudaMemset(); ▷ Memoria en ceros para propagación de modelo real.
    for $it = 0, \cdots, it < nt$ **do**
      cudaMemcpyAsync(Stream1); ▷ Copia de información del *Device* al *host* por medio de *stream*.
      Derivates and CPML zone calculations(Stream2) ▷ Se calculan las derivadas necesarias para la onda P y la zona CPML.
      Field model propagation(Stream0). ▷ Propagación del modelo adquirido en campo con *kernel bloqueante*.
    end for
    Launching Residual Kernel ▷ Se halla el residual.
    $\phi = 0;$ ▷ Variable para la función de costo.
    for $j = 0, \cdots, j < nx * nt$ **do**
      $|residual|_2^2$ ▷ Normal l2 del residual para cálculo de ϕ.
    end for
    for $it = 0, \cdots, it < nt$ **do**
      cudaMemcpyAsync(Stream1) ▷ Copia de datos del *host* al *device* siguiendo el flujo del *stream*1
      Backwards propagation(Stream2) ▷ retropropagación con siguiendo el flujo del *stream*2.
      Gradient calculation(Stream0) ▷ Cálculo del gradiente. Se corre con el *kernel bloqueante*.
    end for
  end for
  for $i = 0, \cdots, i =$ **do**
    Gradient normalization ▷ Normalizar el gradiente.
  end for
  Model Update ▷ Actualización del modelo.
end for

5.3 IMPLEMENTACIÓN DE LA INTERFAZ DE PASO DE MENSAJES

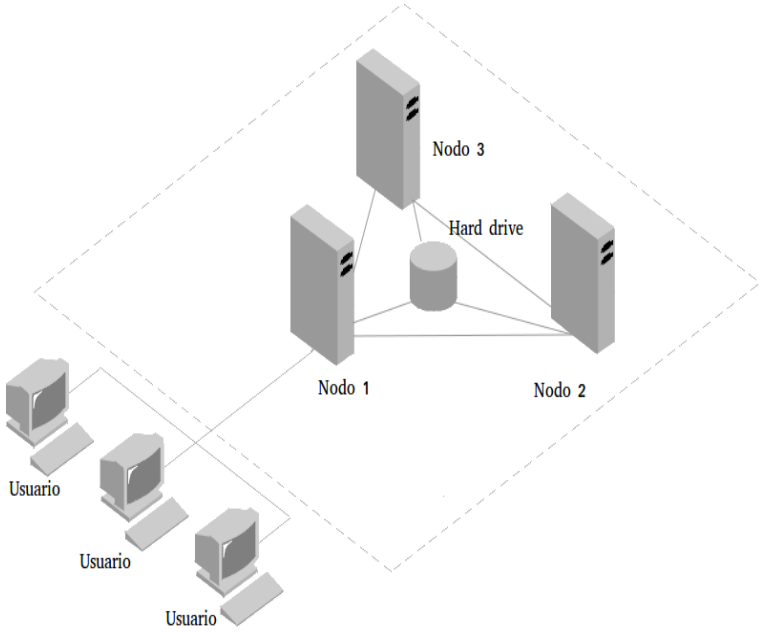
El estándar MPI del inglés *Message Passing Interface* define una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores [18]. El paso de mensajes es una técnica para crear y ejecutar programas que pudieran ser migrados a diferentes computadores conectados en paralelo o nodos [8]. MPI funciona usando un modelo de memoria distribuida, es decir gestiona los recursos del procesador para crear procesos independientes compartiendo los recursos de memoria física, pero teniendo su propia organización de punteros [1].

Con la interfaz se pretende explotar los múltiples procesadores, en este caso las múltiples GPU disponibles para el desarrollo del proyecto. Teniendo en cuenta la cantidad de operaciones que requiere la FWI, utilizar la técnica de MPI resulta de gran ayuda para mejorar el tiempo total del algoritmo usando el concepto de paralelismo para lanzar múltiples fuentes de propagación desde cada nodo.

Aplicar MPI a la inversión realizada aporta a la reducción de tiempos de ejecución del algoritmo al hacer uso de múltiples GPUs, de tal manera que la carga computacional se reparte entre las GPUs del clúster.
El protocolo es viable para su aplicación debido a que para el calculo del gradiente total, no es importante el orden en que se hacen los aportes por disparo. Del mismo modo, la agrupación de dichos aportes por disparo tampoco afectan al resultado total del gradiente.
MPI toma ventaja de lo anterior para dividir la carga computacional entre las múltiples GPUs, haciendo grupos de disparos distribuidos equitativamente entre los nodos dispuestos dentro del clúster. De esta manera cada nodo se encarga de calcular los aportes de este grupo de disparos y acumularlos, para luego sumarlos con el resto de gradientes parciales obtenidos por los otros nodos. Se da la descripción de que es un nodo en la Figura 13.

Se realiza la división equitativa de los disparos en función de las GPUs disponibles. En este caso se dividen por segmentos continuos de disparos, es decir, si se tuvie-

Figura 13: Descripción de un nodo dentro de un clúster.

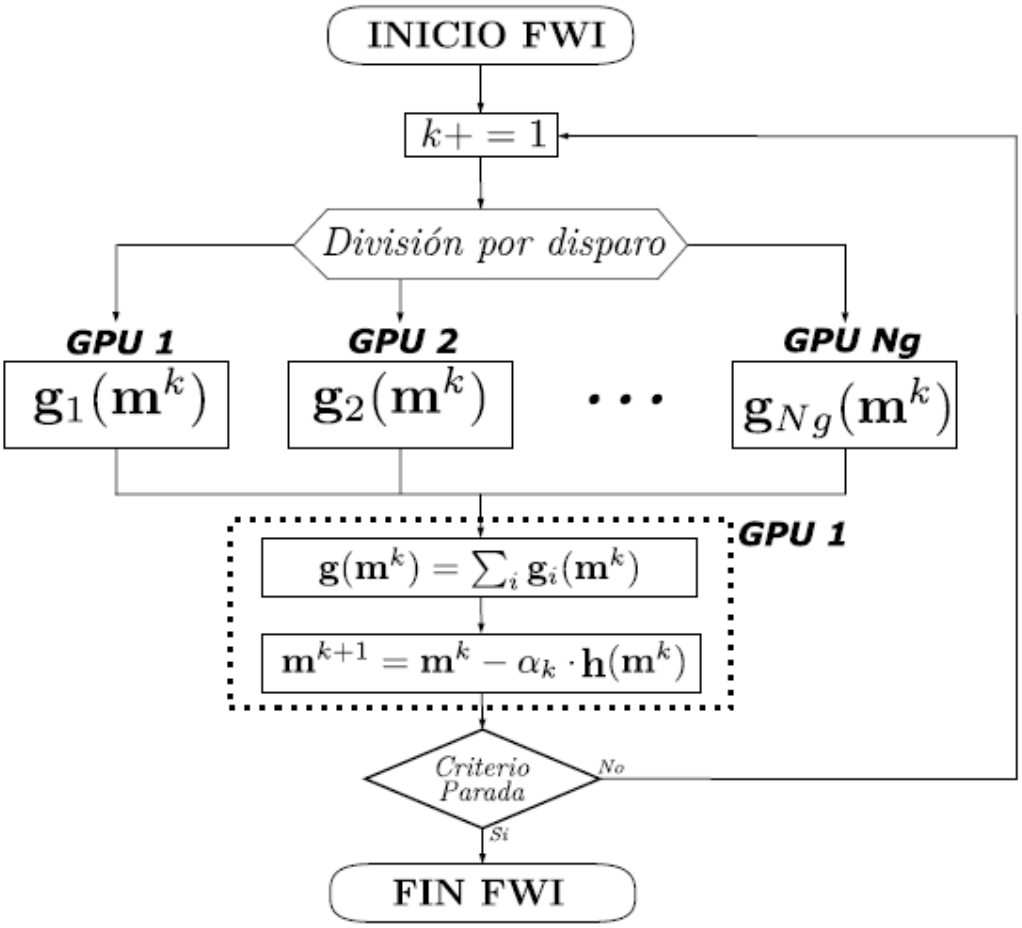


ran 20 disparos y 3 GPUs, la primera GPU realizara el proceso para los primeros 7 disparos, la segunda GPU para los 7 disparos siguientes y la ultima GPU para los 6 disparos restantes. Una vez definidos los disparos por GPU se envía la información necesaria a cada una de ellas, como los modelos de velocidad y la posición de la fuentes.

Cada una de estas GPUs se encargará de calcular el gradiente acumulado para los disparos asignados y una vez se finaliza el calculo de los gradientes parciales, estos son enviados a la GPU designada que suma estos gradientes para obtener el gradiente final. Una vez hecho esto, se actualiza el modelo de velocidades m^{k+1}.

Al utilizar el gradiente descendente como método para la minimización de la función de costo, luego del calculo del gradiente, se determina el paso de actualización del modelo.
Cuando se usa el método L-BFGS, la GPU designada anteriormente, calcula el valor r_k y se realiza la actualización del modelo. Esto agrega un paso adicional para

Figura 14: Flujograma del desarrollo de la FWI con MPI. Imagen tomada de [1]



verificar el cumplimiento de la condición de la Ecuación 50.

Esta propagación adicional se realiza usando la división de disparos. El modelo actualizado m^{k+1} es luego distribuido a todos los nodos para realizar el proceso nuevamente si no se ha cumplido el criterio de parada de la inversión. El esquema general de la implementación de MPI en la FWI se muestra en la Figura 14.

6. RESULTADOS Y DISCUSIÓN

En este capitulo se hace la validación del algoritmo tomando como referencia tres modelos a los cuales se les realiza el proceso de inversión. También se hace una comparación entre las estrategias computacionales planteadas, teniendo presente dos factores computacionales los cuales son RAM ocupada en la GPU y tiempo de ejecución.

El algoritmo se valida a partir de la inversión de 3 modelos, cada uno con diferentes características físicas. Una breve descripción de cada modelo a invertir se dará en la sección 6.1. Esto con el fin de exponer algunas de las ventajas y desventajas de usar la inversión de onda completa con anisotropía VTI. Se eligen modelos con dimensiones diferentes para poner a prueba las estrategias en diferentes condiciones.

Las pruebas de inversión expuestas en esta sección se ven ordenadas de menor a mayor exigencia computacional. Se exponen los tiempos de ejecución y la memoria requerida de ambos algoritmos. Se parte de un modelo inicial con diferentes características de acuerdo a los requerimientos que las estrategias demanden para cada modelo. Se dan las mismas condiciones computacionales al iniciar la inversión de un modelo con las diferentes estrategias implementadas.

Para cada modelo de validación del algoritmo, al finalizar la FWI, se obtienen tres coeficientes de velocidad. Las velocidades de la onda P dadas en la ecuación 2 se presentarán en conjuntos de tres imágenes por cada coeficiente de velocidad, cada uno de estos conjuntos representando el modelo inicial, el modelo invertido y el modelo sintético.

El modelo inicial es el punto de partida del algoritmo, a partir de este se hace la inversión, este modelo define en gran medida el modelo invertido. El punto de partida es muy importante en el descenso del gradiente ya que este método se basa en el criterio de la primera derivada para resolver un problema mediante optimización.

Este criterio funciona muy bien dentro de intervalos cerrados. Para intervalos abiertos el problema es más complejo. Con intervalos abiertos se da cabida a los mínimos locales y debido al criterio de parada del algoritmo, se puede asumir que un mínimo local es la solución.
El modelo invertido es la solución a la cual se llegó a partir del modelo inicial.
Es el modelo actualizado resultante de la inversión a $3Hz$, $6Hz$ y $9Hz$ producto del multiescalado que se implementó en el algoritmo.

El modelo sintético es el punto a donde el algoritmo debería llegar. Este modelo se obtiene de un software de análisis geosísmico. El modelo sintético participa en el cálculo del gradiente y en el descenso de función de costo, puesto que el método busca llevar el error entre el modelo inicial y el modelo sintético a un mínimo relativo o absoluto. Lo anterior con la norma $l2$ al cuadrado de la diferencia entre los *shot gather* de la propagación de este modelo y el modelo inicial.

Se hace un análisis de la relación señal a ruido de pico o PSNR, por sus siglas en inglés, para determinar la calidad del resultado de la inversión con respecto al modelo original. El PSNR se calcula con la ecuación

$$PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right), \tag{58}$$

donde MAX es la velocidad máxima y MSE es la diferencia al cuadrado entre los valores de velocidad del modelo original y el modelo invertido. Para valores mayores de PSNR se tiene una mayor calidad en la imagen con respecto al modelo original.

6.1 MÉTODO DEL PUNTO DE REFERENCIA

En el método de *checkpointing* pese a usar los mismos *kernels* que el método *hidden latency*, estos *kernels* se aplican de forma diferente. El método de *checkpointing* secciona la propagación junto con la retropropagación para hacer el cálculo del gradiente por lo que cambian las funciones de costo y el resultado de la inversión. La cantidad de *checkpoitns* afecta al resultado de la FWI generando errores en datos de propagación.

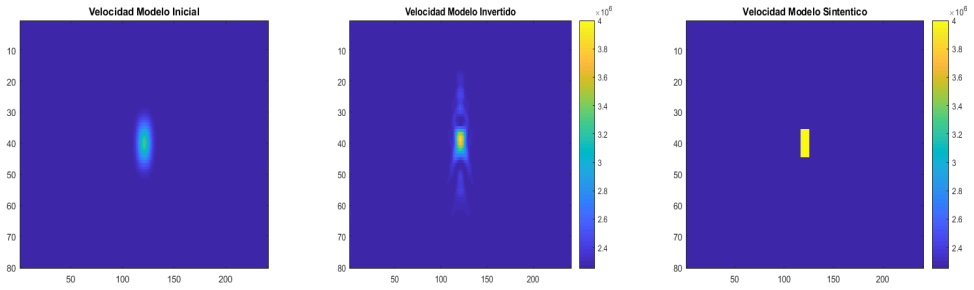
Para que esta estrategia funcione de manera correcta es necesario cumplir ciertas condiciones. Dentro de la sección 5.1 se definió el valor óptimo de *checkpoints* a guardar para tener la menor cantidad de memoria reservada dado en la ecuación 57. De esta manera la suma de memoria para los estados guardados y la memoria para el volumen de datos de la propagación es menor. Debido a la forma misma de la estrategia, a mayor cantidad de *checkpoints*, se deben propagar mayor cantidad de segmentos. Al truncar los datos en cada segmento propagado, se generan los errores numéricos en los datos de propagación y retropropagación y esto conlleva a un error en el cálculo del gradiente. Debido a lo anterior es mejor y mas óptimo definir una cantidad de *checkpoints* menor.

Esta estrategia tiene varias desventajas frente al método de *hidden latency*. La principal desventaja es el error numérico asociado a la cantidad de puntos guardados durante la propagación. Este error crece de forma directa a la cantidad de *chekpoints* guardados lo que genera un comportamiento erróneo en el gradiente y en el método del L-BFGS.

6.1.1 Modelo del cuadrado difractor Este modelo es el mas simple de los modelos propuestos para la validación de las estrategias. El modelo consiste en una capa constante de velocidad, dicha capa tiene un cuadrado con un mayor valor de velocidad en el centro. Este patrón se replica de la misma manera en los tres parámetros a invertir. Debido a las implicaciones físicas del modelo, se resaltan algunas de las fallas de la FWI para modelos con parámetros de anisotropía VTI.

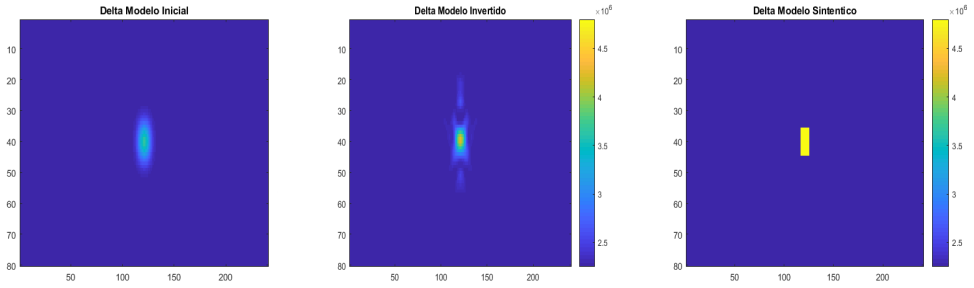
Las falencias de la FWI se dan en sombras que aparecen sobre el modelo y una pérdida de resolución en las figuras alargadas. El problema de las sombras genera un adelgazamiento en las partes mas bajas del modelo, esto se ve con mayor claridad en los parámetros invertidos del cuadrado en las figuras 15, 16 y 17. Las sombras también generan algunos ruidos en la parte baja del cuadrado.

Figura 15: Modelo de velocidad c_z obtenido para el modelo del cuadrado difractor con estrategia del *checkpointing*.



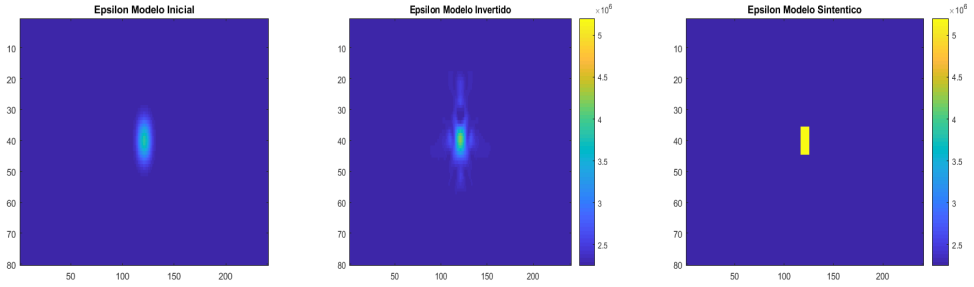
(a) Velocidad c_Z inicial. (b) Velocidad c_Z Invertida. (c) Velocidad c_Z Original.

Figura 16: Modelo de velocidad d_x obtenido para el modelo del cuadrado difractor con estrategia del *checkpointing*.



(a) Velocidad inicial. (b) Velocidad d_x Invertida. (c) Velocidad d_x Original.

Figura 17: Modelo de velocidad c_x obtenido para el modelo del cuadrado difractor con estrategia del *checkpointing*.



(a) Velocidad c_x inicial. (b) Velocidad c_x Invertida. (c) Velocidad c_x Original.

El modelo del cuadrado, invertido mediante la estrategia de *checkpointig* tiene varias fallas por fuera de las causadas por la FWI para modelos con anisotropía VTI como las sombras expuestas anteriormente. Estas fallas debido a la forma en que se aplican los diferentes *kernels* que propagan.
Observando la escala de colores y la definición del cuadrado es claro que las condiciones establecidas en L-BFGS rompen con la inversión antes de que esta puede llegar al mínimo local. Esto debido a que dentro de las condiciones dadas para aplicar L-BFGS, se deja de hacer más iteraciones una vez que el error deja de decrecer. Por lo tanto, se obtiene una inversión con magnitudes menores y con carencia de definición. Pese a eso hubo una mejora respecto al modelo inicial.

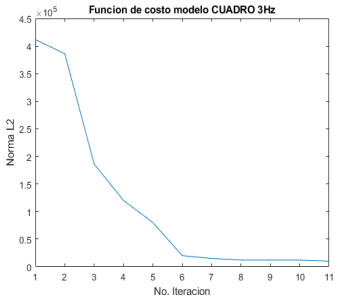
El valor de PSNR calculado para cada uno de las velocidades fueron de $32dB$, $34{,}2dB$ y $35{,}5dB$ para las velocidades c_x, d_x y c_z respectivamente.

El comportamiento en cuanto al decrecimiento del error se puede ver con mayor claridad en las funciones de costo presentadas en la figura 18. De los tres modelos el cuadrado fue el que tuvo un mejor comportamiento en cuanto a decrecimiento del error, siendo la frecuencia de $6\ hz$ la que tuvo una reducción mayor respecto a su valor inicial.

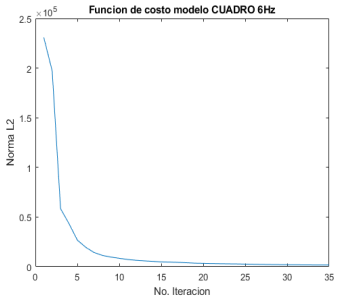
Para cada inversión hecha a 3, 6 y $9\ Hz$ se obtiene una gráfica de la función de costo por cada frecuencia, mostrada en la figura 18

Estas gráficas de las funciones de costo muestran cuantas iteraciones se necesitaron para llegar al valor mínimo de la función de costo para cada inversión. Se puede observar como para las inversiones a 3 y $6\ Hz$ se requieren entre cinco y seis iteraciones para lograr un error menor al $0{,}5 * 10^5$. Para concretar los detalles del modelo con los componentes frecuenciales altos se requieren más iteraciones del algoritmo.

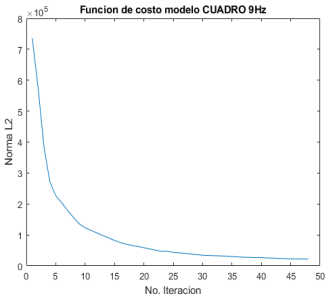
Figura 18: Funciones de costo para multifrecuencia con el modelo del cuadrado difractor con *checkpointing*.



(a) Función de costo a 3 Hz para el modelo del cuadrado.



(b) Función de costo a 6 Hz para el modelo del cuadrado.



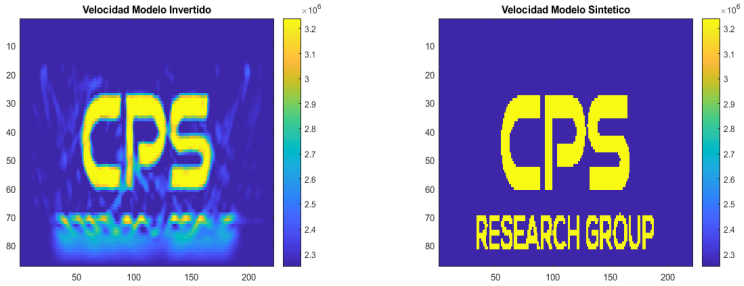
(c) Función de costo a 9 Hz para el modelo del cuadrado.

También se puede ver como el método L-BFGS hace más pronunciada la pendiente a partir de la segunda iteración, cumple con los requisitos estipulados para la aplicación de este. Para notar un buen funcionamiento de L-BFGS, debería verse una disminución del error entre dos veces el punto de error dado por el método del gradiente y una década de descenso.

6.1.2 Modelo logo de CPS El modelo del logo de CPS es muy similar al del cuadrado con la distinción de tener el logotipo del grupo de investigación CPS en el centro. La particularidad de estos dos modelos ya mostrados es que el patrón costa solo de dos variaciones en los parámetros. Al igual que en el modelo del cuadrado las zonas con mayor dificultad para invertir son las alargadas. Se pueden notar en las figuras 19, 20 y 21 algunos adelgazamientos en las zonas más bajas. En la P

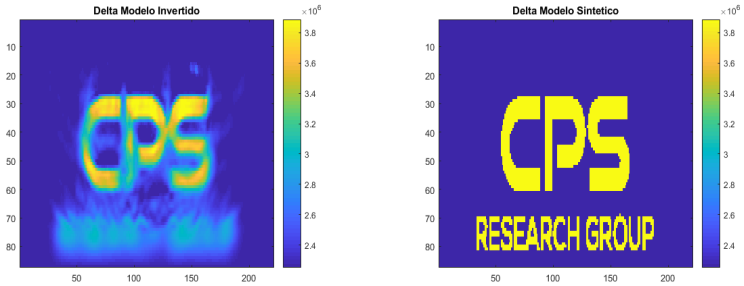
del logotipo este factor se ve fuertemente potenciado. Una de las fallas que mas destaca en este modelo es la dispersión numérica, producto del paso temporal en el proceso de propagación hacia adelante y hacia atrás.

Figura 19: Modelo de velocidad c_z obtenido para el modelo CPS con estrategia del *checkpointing*.



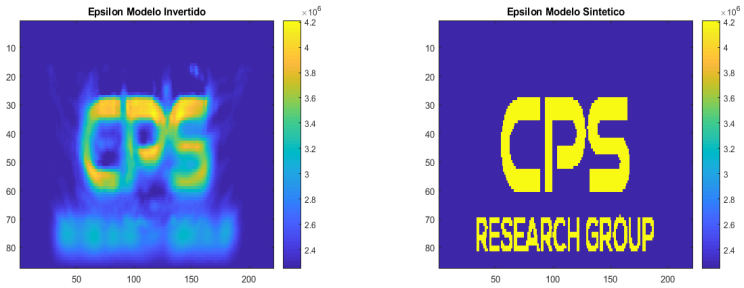
(a) Velocidad c_z Invertida. (b) Velocidad c_z Original.

Figura 20: Modelo de velocidad d_x obtenido para el modelo CPS con estrategia del *checkpointing*.



(a) Velocidad d_x Invertida. (b) Velocidad d_x Original.

Figura 21: Modelo de velocidad c_x obtenido para el modelo CPS con estrategia del *checkpointing*.



(a) Velocidad c_x Invertida. (b) Velocidad c_x Original.

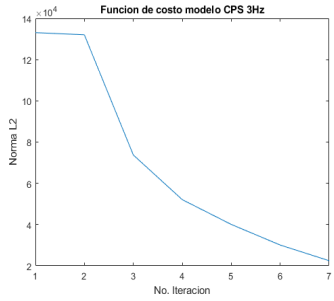
Para el modelo CPS el comportamiento de decrecimiento del error fue menor en comparación con el cuadrado, al igual que el número de iteraciones. Esto es un indicativo de que el error no decrece y debido a las condiciones de parado de L-BFGS detiene la inversión y guarda los modelos hasta donde llegaron en el descenso del error. Observando las escalas de colores en comparación con el modelo sintético se puede concluir que la inversión genera la forma del modelo. A pesar de generar la forma del modelo, no se llega a los niveles de saturación de este, lo que a nivel físico es un error importante en el uso de los datos.

El valor de PSNR calculado para cada uno de las velocidades fueron de $19{,}6dB$, $20{,}9dB$ y $23{,}9dB$ para las velocidades c_x, d_x y c_z respectivamente.

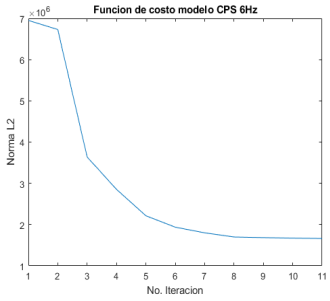
Las funciones de costo para el modelo del logo de CPS con multiescala se ven en la figura 22. Se puede apreciar que para el algoritmo es más difícil hacer descender el error. Esto puede ser causado por la dificultad que tiene el modelo a invertir, dado el alto contraste entre zonas con diferentes valores. El método de L-BFGS también hace su aparición en este modelo haciendo que el error que es casi lineal para las dos primeras iteraciones tenga una bajada de casi el doble que el logrado por el método del descenso del gradiente. Es posible que el error alcanzado no sea tan pequeño debido a las condiciones de parado que tiene L-BFGS, donde una

vez el error deja de descender sustancialmente, detiene el algoritmo terminando la inversión.

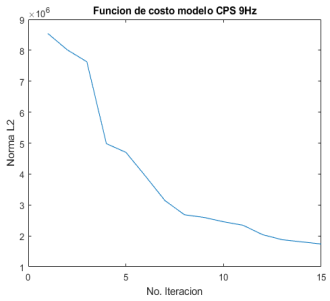
Figura 22: Funciones de costo para multifrecuencia con el modelo del CPS con *checkpointing*.



(a) Función de costo a 3 Hz para el modelo de CPS.



(b) Función de costo a 6 Hz para el modelo de CPS.



(c) Función de costo a 9 Hz para el modelo de CPS.

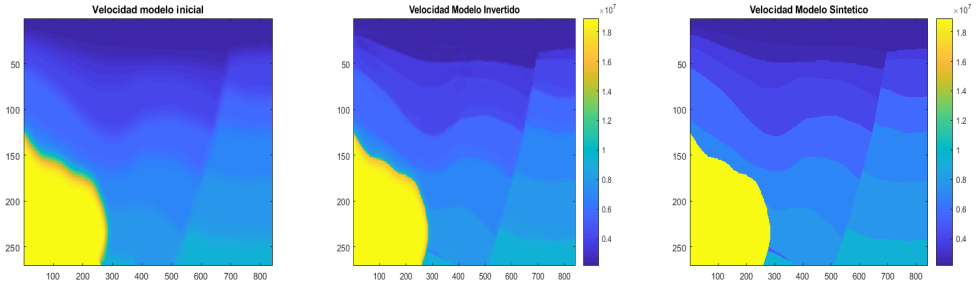
6.1.3 Modelo Hess El modelo Hess es el modelo más complejo físicamente de entre los tres modelos. Este modelo es una porción del modelo original.

Difiere bastante de los modelos del cuadrado y de CPS ya que este consta de un patrón muy diferente. En este modelo hay presencia de múltiples capas para cada parámetro. Cabe aclarar que en este modelo las velocidades c_x y d_x no son una replica del modelo de velocidad c_z, que es una característica de los modelos del cuadrado y del logo de CPS. Otra distinción destacable es la zona de alta velocidad que tiene el modelo en la parte baja.

Las zonas de alta velocidad pueden generar comportamientos erróneos en la inversión. Estas zonas generan un contraste muy alto de energía durante la propagación

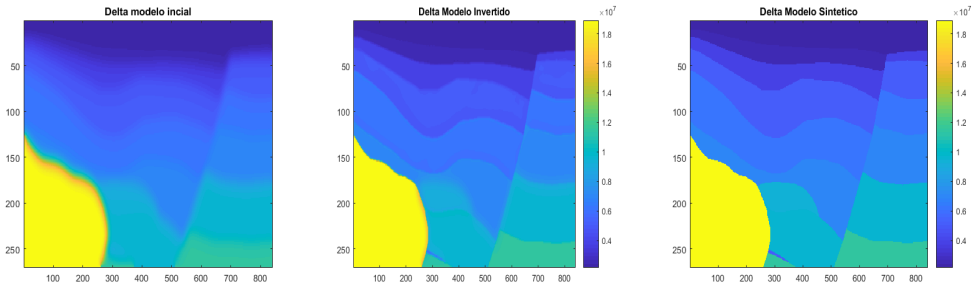
generando el efecto de un prisma en las partes mas cercanas a la zona de alta velocidad y produciendo un comportamiento distintivo en la función de costo.

Figura 23: Modelo de velocidad c_z obtenido para Hess con estrategia del *checkpointing*.



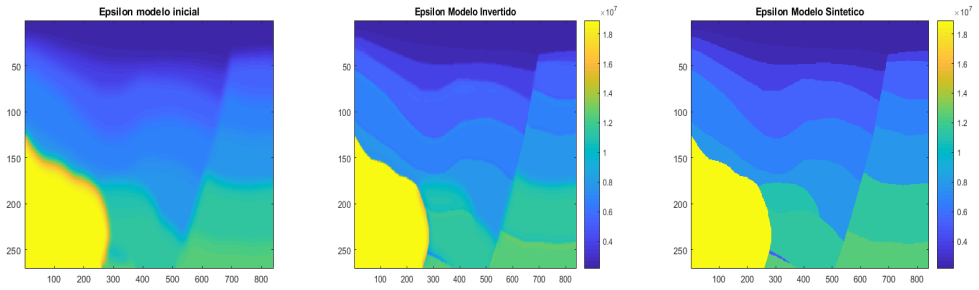
(a) Velocidad c_z inicial. (b) Velocidad c_z Invertida. (c) Velocidad c_z Original.

Figura 24: Modelo de velocidad d_x obtenido para Hess con estrategia del *checkpointing*.



(a) Velocidad d_x inicial. (b) Velocidad d_x Invertida. (c) Velocidad d_x Original.

Figura 25: Modelo de velocidad c_x obtenido para Hess con estrategia del *checkpointing*.



(a) Velocidad c_x inicial. (b) Velocidad c_x Invertida. (c) Velocidad c_x Original.

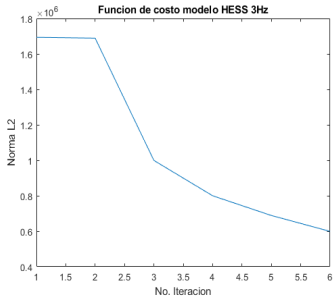
El modelo de Hess fue el más complejo de invertir con esta estrategia puesto que el descenso del error fue muy poco. Hubo cambios respecto al modelo inicial pero no de forma destacable. Esto se puede ver con más claridad al observar el descenso de la función de costo. El descenso de la función es muy lento y en algunas zonas de la gráfica es parcialmente lineal.

El valor de PSNR calculado para cada uno de las velocidades fueron de $35{,}2dB$, $36{,}6dB$ y $38{,}9dB$ para las velocidades c_x, d_x y c_z respectivamente.

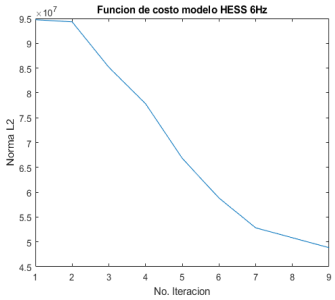
Las funciones de costo en la figura 26 para el modelo Hess, a pesar de tener un error final menor al logrado con el modelo CPS requiere más iteraciones para lograrlo.

La función de costo empieza con un error relativamente bajo en comparación con los anteriores modelos. A pesar de esto, se ve como requiere más iteraciones para lograr un error aceptable. Las funciones de costo no son pronunciadas a pesar de que el método L-BFGS hace que el error descienda a un buen paso. El método del descenso del gradiente produce una disminución del error muy lenta llegando a ser casi nula en las primeras iteraciones del algoritmo.

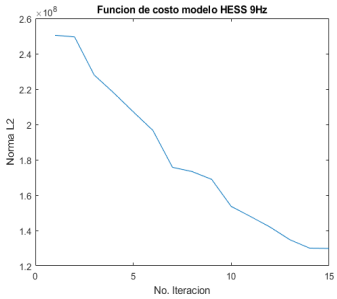
Figura 26: Funciones de costo para multifrecuencia con el modelo del Hess con *checkpointing*.



(a) Función de costo a 3 Hz para el modelo Hess.



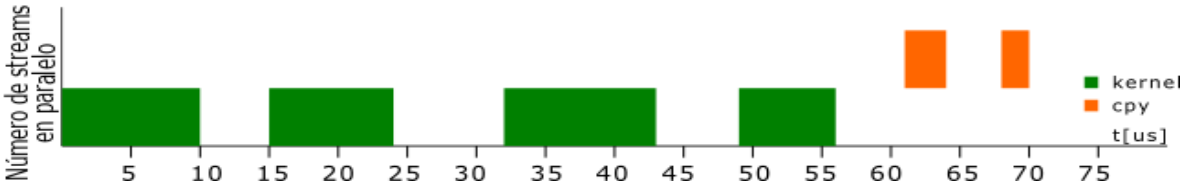
(b) Función de costo a 6 Hz para el modelo Hess.



(c) Función de costo a 9 Hz para el modelo Hess.

Las líneas de tiempo 27 y 28, representan el flujo de ejecución de los *kernels* para propagación y retropropagación. De esta manera se puede ver cuáles de los *kernels* tienen más exigencia computacional basados en sus tiempos de ejecución. A diferencia de la estrategia *hidden latency* esta estrategia no utiliza *streams* por tanto los tiempos de latencia entre cada *kernel* es menor, esto debido a que solo existe un contexto de ejecución. Otro punto a favor del método del *checkpointing* es que la propagación y retropropagación operan únicamente en la GPU. Es decir, los datos permanecen siempre en la GPU mientras que el método *hidden latency* copia los datos al *host*.

Figura 27: Linea de tiempo para la propagación en *checkpointing*.

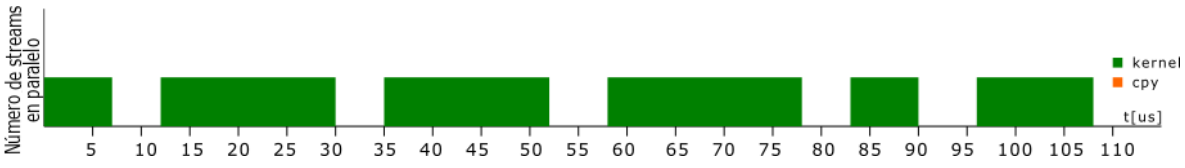


Cuadro 1: *Kernels* y tiempos de ejecución para la propagación en *checkpointing*.

Proceso de Propagación Hacia Adelante			
Kernels	Inicio de Ejecución[us]	Final de Ejecución[us]	Duración de Ejecución[us]
PSI	0	10	10,048
Segunda Derivada	15	24,228	9,728
ZETA	32	43,265	11,265
Propagación	49	56,262	7,262
Cpy[DtD]	61	63,140	2,14
Cpy[DtD]	68	70,120	2,12

El cuadro 1 describe los *kernels* usados en la implementación de la propagación de esta estrategia, da detalles sobre los tiempos de inicio, finalización y ejecución de cada *kernel*. Se puede ver que el *kernel* con más carga es el *kernel* llamado ZETA, encargado de calcular la variable auxiliar para las fronteras CPML dada en la ecuación 4

Figura 28: Linea de tiempo para la retropropagación en *checkpointing*.



Cuadro 2: *Kernels* y tiempos de ejecución para la retropropagación en *checkpointing*.

Proces de Propagación Hacia Atras			
	Inicio de Ejecución[us]	Final de Ejecución[us]	Duración de Ejecución[us]
Punto de Entrada	0	6,624	6,624
Segunda Derivada	12	29,504	17,504
PSI	35	52,728	17,728
ZETA	58	78,48	20,48
Retro-Propagación	83	90,712	7,712
Gradiente	96	108,64	12,640

El cuadro 2 da detalles sobre los tiempos que toma cada *kernel* usado en la retropropagación con la estrategia del *checkpointing*. Estos *kernels* tienen mayor carga computacional dado que para la retropropagación se utilizan las ecuaciones obtenidas con el método del estado adjunto dadas en 32. Estas ecuaciones requieren mayor número de operaciones para su solución.

La forma de segmentar el algoritmo genera un mayor tiempo de ejecución ya que agrega una propagación más al proceso. El algoritmo de la FWI está basado en tres pasos por cada disparo de fuente. El primero es la propagación con el modelo sintético, con el propósito de guardar los *shot gather*. A continuación, se hace la propagación del modelo inicial que en la siguiente iteración se reemplaza por el modelo actualizado. Después de dicha propagación es necesario guardar los *shot gather* para hallar la función de costo. Hasta este punto el algoritmo se implementa de la misma forma que la estrategia *hidden lantecy*. Pero ya que solo se reservó un volumen parcial como se explica en la sección 5.1, es necesario propagar durante el proceso de retropropagación para hallar los gradientes y aplicar el método del descenso del gradiente.

Esto agrega una propagación más al flujo de la FWI, pero esta propagación extra siempre será la misma independiente del número de *checkpoints* asignados.

Este tiempo extra que ocupa el algoritmo se podrá ver con mayor claridad en los cuadros 3 y 4.

Los cuadros muestran la cantidad de llamados que se hacen por *kernel*. En comparación con la estrategia de *hidden latency* en *checkpointing* se hace $1/3$ más de llamados tomando como referencia la estrategia tradicional (que es operar la FWI en su totalidad dentro de la GPU sin aplicar segmentación). El método del *checkpointing* puede tener muy buenos resultados en cuanto a la reducción de la memoria utilizada. A pesar de reducir la memoria utilizada, el método tiene fallos numéricos explicados anteriormente.

Pese a tener latencias muy bajas producto de los contextos de ejecución, el hecho de tener que hacer otra propagación genera mayores tiempos de ejecución.

Cuadro 3: Carga Computacional para la estrategia *Checkpointing*

Distribución de Carga Computacional		
Operación	Tiempo[%]	No. Llamadas
Propagador	24.18 %	15000
SegundaDerivada_P	20.23 %	15000
SegundaDerivada_Bp	11.01 %	5000
Gradiente	9.12 %	5000
Propagador_HaciaAtrás	8.04 %	5000
Zeta_P	7.57 %	15000
Psi_P	6.63 %	15000
PuntoDeEntrada	5.09 %	5000
Zeta_BP	3.53 %	5000
Psi_BP	3.15 %	5000
MemCpy	1.45 %	10032

Cuadro 4: Operaciones realizadas por la estrategia *Checkpointing*

Operaciones del Sistema		
Operación	Tiempo[%]	No. Llamadas
kernels	78.85 %	90008
CudaMemSet	11.39 %	103
CudaMemCpy	4.75 %	10044
Llamadas al Sistema	4.65 %	8013
CudaMalloc	0.36 %	46

En los cuadros 3 y 4 se puede ver como está distribuida la carga computacional para cada paso del algoritmo. Las transferencias de datos dentro de esta estrategia son en su mayoría operadas dentro de la GPU, por lo tanto, no se generan retardos producto de la transferencia de datos.

Las operaciones del sistema dadas en el cuadrado 4 hacen referencia al llamado de los *kernels* descritos, las reservas de memoria con el comando *CudaMalloc*, el establecimiento de un valor para la memoria con instrucción *CudaMemSet* y las copias de información con el comando *CudadMemCpy*.

Durante las inversiones se tomaron los datos en cuanto a memoria ocupada y tiempo de ejecución. Para 4 segmentaciones del algoritmo se obtuvieron, en el cuadro 5 los resultados del modelo del cuadrado difractor, en el cuadro 6 los resultados para el modelo del logo de CPS y finalmente en el cuadro 7 los resultado para el modelo Hess.

Cuadro 5: Memoria y tiempo de ejecución para el modelo del cuadrado difractor en *checkpointing*.

Memoria en la GPU	Tiempo de ejecución del algoritmo
161[Mb]	$68{,}36$[s]

Cuadro 6: Memoria y tiempo de ejecución para el modelo del logo de CPS en *checkpointing*.

Memoria en la GPU	Tiempo de ejecución del algoritmo
159[Mb]	$69{,}32$[s]

Cuadro 7: Memoria y tiempo de ejecución para el modelo de Hess en *checkpointing*.

Memoria en la GPU	Tiempo de ejecución del algoritmo
1380[Mb]	$168,32$[s]

6.2 MÉTODO DE LATENCIA OCULTA

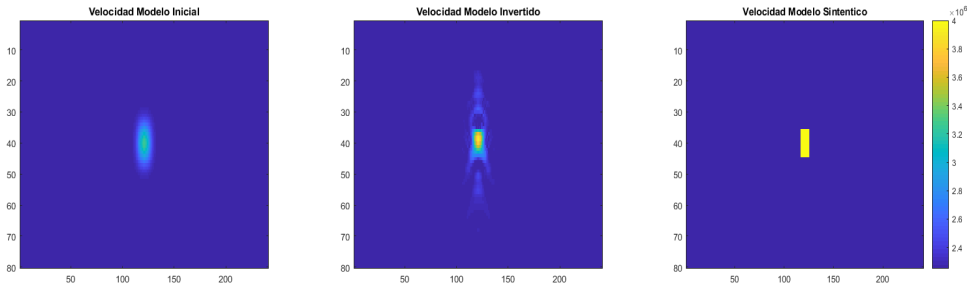
El método de *hidden lantecy* se basa completamente en la estructura básica de la inversión donde se respeta por completo la forma en la que se aplica el algoritmo. Esta estrategia aprovecha las funciones de Cuda y la posibilidad de crear hilos de ejecución dentro del *host* para lanzar los *kernels* y las transferencias de datos de forma simultánea. Esto trae grandes ventajas, ya que los cálculos de la propagación se hacen solo una vez y dado que la transferencia de datos no se corrompe, los errores numéricos producto de cálculos computacionales se ven reducidos.
La transferencia de datos no introduce errores puesto que los datos se transfieren de forma integra y sincronizada. El método hace que la RAM del *host* sea una extensión de la RAM de las GPUs, por tanto hay mayor cantidad de memoria a disposición de las los procesadores gráficos.

Los defectos principales de este método se ven reflejados en el lanzamiento de los *kernels*, ya que dependiendo de la cantidad de GPUs por nodo, la cantidad de *streams* crecen dentro del mismo. Una mayor cantidad de *streams* genera un retardo entre lanzamientos de *kernels*. Esto es producto de la forma en la que el sistema gestiona los *streams*.

Algo a tener en cuenta es que este método se limita por la cantidad de RAM del *host*. El uso de MPI reserva gran cantidad memoria dentro del *host*. Puesto que las reservas de memoria se replican para cada proceso generado por la aplicación de MPI, la RAM del *host* puede limitar la cantidad de memoria usada para el desarrollo de la FWI .

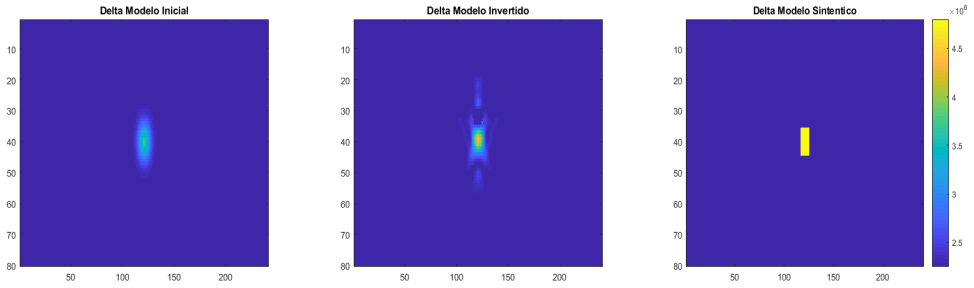
6.2.1 Modelo del cuadrado difractor En la sección de *checkpointing* se discutieron las fallas generales de la inversión de este modelo producto de los errores de hacer una inversión con parámetros de anisotropía VTI. Véase la sección 6.1.1.

Figura 29: Modelo de velocidad c_z obtenido para el modelo del cuadrado difractor con la estrategia *hidden latency*.



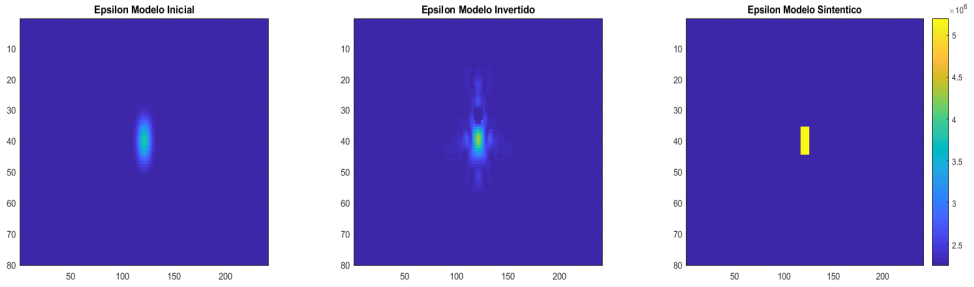
(a) Velocidad c_z inicial. (b) Velocidad c_z Invertida. (c) Velocidad c_z Original.

Figura 30: Modelo de velocidad d_x obtenido para el modelo del cuadrado difractor con la estrategia *hidden latency*.



(a) Velocidad d_x inicial. (b) Velocidad d_x Invertida. (c) Velocidad d_x Original.

Figura 31: Modelo de velocidad c_x obtenido para el modelo del cuadrado difractor con la estrategia *hidden latency*.



(a) Velocidad c_x inicial. (b) Velocidad c_x Invertida. (c) Velocidad c_x Original.

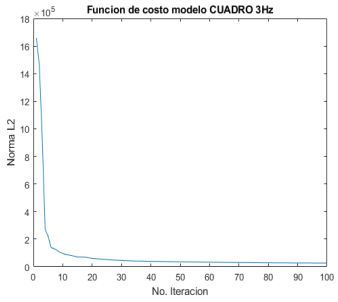
El modelo del cuadrado difractor con la estrategia *hidden latency* tiene un buen comportamiento y un descenso del error bastante bueno. El modelo presenta algunas variaciones respecto al modelo sintético debido las fallas de la FWI. Puesto que al tener en cuenta solo los parámetros verticales, se producen algunas sombras sobre el modelo. La definición del cuadrado en las zonas superiores e inferiores no es muy alta pero en las zonas laterales del cuadrado tiene una mejor definición. Si se observan el rango de colores en los modelos inicial e invertido, es posible ver como el modelo invertido alcanza las magnitudes del modelo sintético.

Respecto a la forma y definición del modelo, se ven afectadas debido a las fallas naturales del algoritmo, esto se ve con mayor magnitud en los modelos de solo dos capas. Esta falla se ve con mejorías en el modelo Hess. Esto puede ser producto de la variación brusca entre las dos capas y debido a la forma del cuadrado.

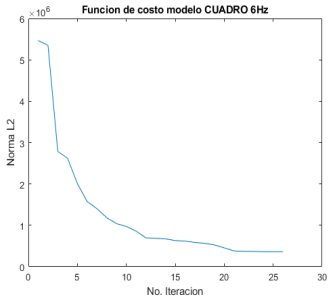
El valor de PSNR calculado para cada uno de las velocidades fueron de $35{,}4dB$, $32{,}9dB$ y $34{,}77dB$ para las velocidades c_x, d_x y c_z respectivamente.

Las funciones de costo para multifrecuencia con el modelo del cuadrado difractor se pueden ver en la figura 32. Se puede ver un muy buen descenso en el error para este modelo. Son necesarias aproximadamente 10 iteraciones del algoritmo para conseguir un error aceptable.

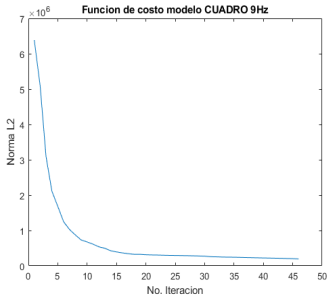
Figura 32: Función de costo para el modelo del cuadrado difractor con multifrecuencia en *hidden latency*.



(a) Función de costo a 3 Hz para el modelo del cuadrado difractor.



(b) Función de costo 6 Hz para el modelo del cuadrado difractor.



(c) Función de costo 9 Hz para el modelo del cuadrado difractor.

6.2.2 Modelo logo de CPS Con la estrategia de *hidden latency* se puede notar a simple vista una mejoría en los resultados a comparación con la estrategia del *checkpointing*. Aunque sigue presentando dificultades para hacer la inversión de las partes más alargadas y bajas del modelo, la parte superior tiene gran detalle y una muy buena definición.

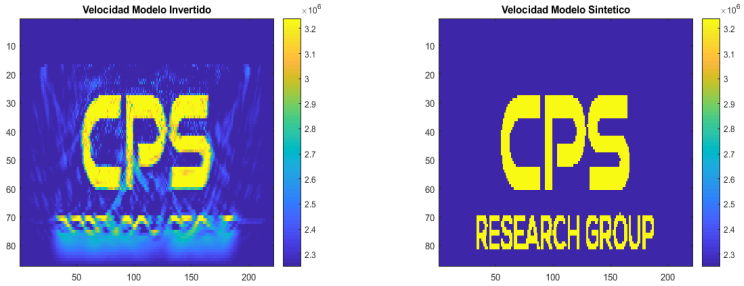
Mirando la escala de colores la inversión de los parámetros es bastante acertada, siendo la velocidad c_x la que presentó mayor dificultad y error al momento de invertir.

Se sigue presentando el fenómeno de las sombras sobre el modelo, como se puede observar en las figuras 34(a) y 35(a). A pesar de estas sombras se logra un resultado muy bueno.

Para la figura 33(a) se puede decir que la inversión hecha a $9\ Hz$ aporta bastantes detalles pues la definición es muy buena, recordando que las altas frecuencias dan detalles al modelo. Para el modelo inicial se tiene una capa constante de velocidad $1500m/s$.

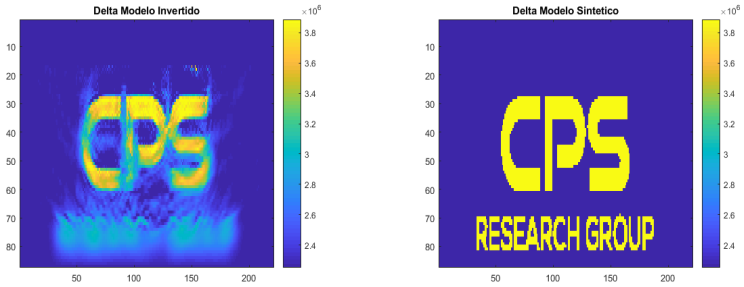
El valor de PSNR calculado para cada uno de las velocidades fueron de $24{,}5dB$, $19{,}6dB$ y $20{,}9dB$ para las velocidades c_x, d_x y c_z respectivamente.

Figura 33: Modelo de velocidad c_z obtenido para el CPS con *hidden latency*.



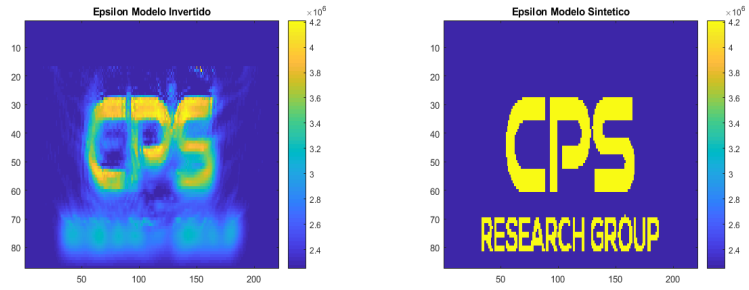
(a) Velocidad c_z Invertida. (b) Velocidad c_z Original.

Figura 34: Modelo de velocidad d_x obtenido para el modelo CPS con *hidden latency*.



(a) Velocidad d_x Invertida. (b) Velocidad d_x Original.

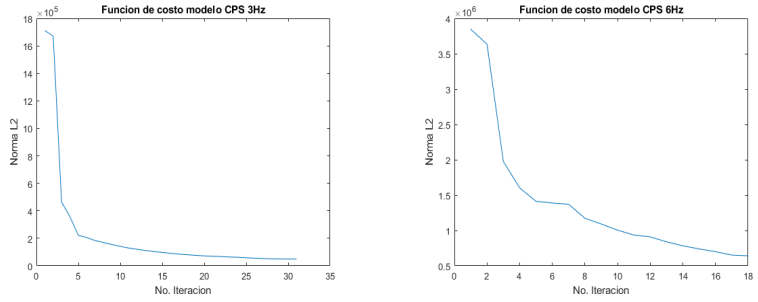
Figura 35: Modelo de velocidad c_x obtenido para el modelo CPS con *hidden latency*.



(a) Velocidad c_x Invertida.

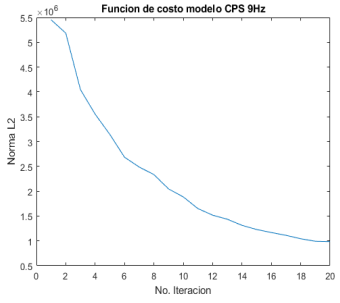
(b) Velocidad c_x Original.

Figura 36: Función de costo para modelo CPS con multifrecuencia en *hidden latency*



(a) Función de costo a 3 Hz para modelo CPS.

(b) Función de costo a 6 Hz para modelo CPS.



(c) Función de costo a 9 Hz para modelo CPS.

La función de costo demuestra que para la inversión a baja frecuencia en la figura

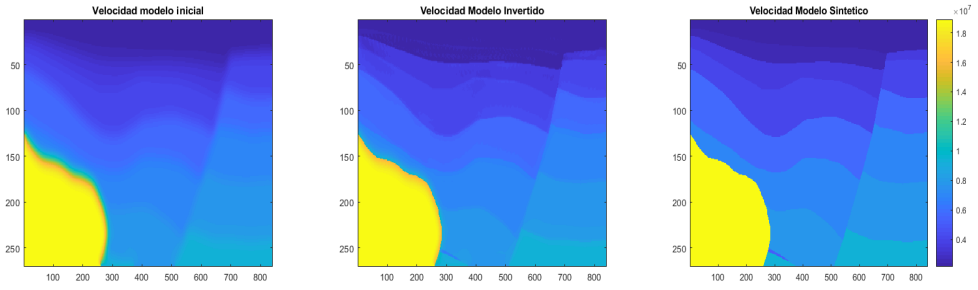
36(a) la disminución del error es sustancial y permite un avance en iteraciones de hasta 30. Para medias y bajas frecuencias en las figuras 36(b) y 36(c) no se permiten tantas iteraciones, esto debido al criterio de parada del método L-BFGS donde el error deja de descender y se detiene la inversión.

6.2.3 Modelo Hess la aplicación de la estrategia de *hidden latency* da muy buenos resultados para grandes cantidades de datos, el modelo Hess es un muy buen ejemplo de esto.

Se puede ver en las figuras 37, 38 y 39 que los resultados de la inversión son muy buenos, se tiene gran detalle del modelo y una muy buena definición de cada capa. Por esto se demuestra que la estrategia hace un muy buen manejo de datos, sin generar corrupción de estos.

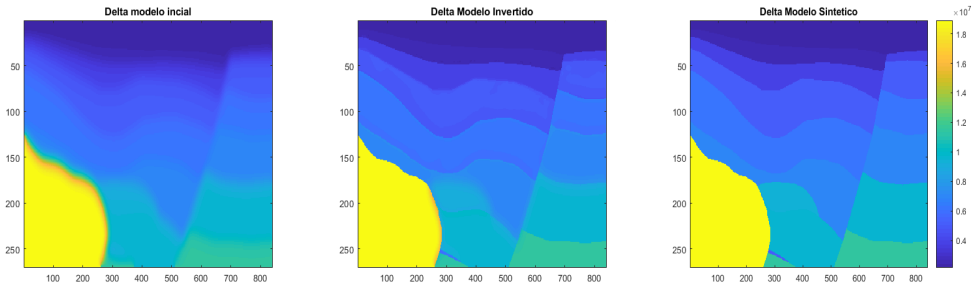
Las características de concurrencia del método para las copias y los lanzamientos de los *kernels* se comportan mejor para grandes cantidades de datos. Por lo que se tiene un mejor desempeño para el modelo Hess en comparación a los otros dos modelos.

Figura 37: Modelo de velocidad c_z obtenido para Hess con *hidden latency*.



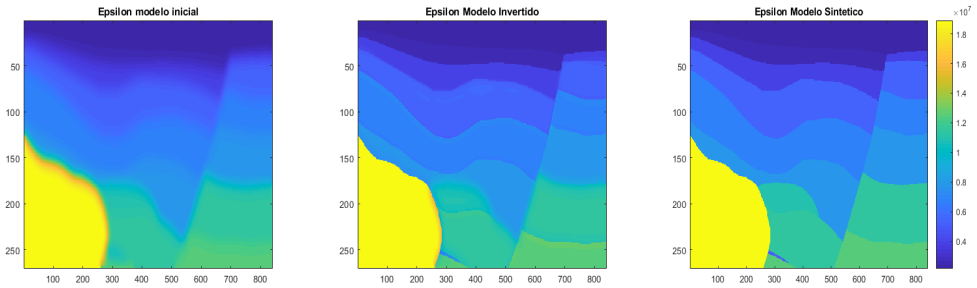
(a) Velocidad c_z inicial. (b) Velocidad c_z Invertida. (c) Velocidad c_z Original.

Figura 38: Modelo de velocidad d_x obtenido para Hess con *hidden latency*.



(a) Velocidad d_x inicial. (b) Velocidad d_x Invertida. (c) Velocidad d_x Original.

Figura 39: Modelo de velocidad c_x obtenido para Hess con *hidden latency*.



(a) Velocidad c_x inicial. (b) Velocidad c_x Invertida. (c) Velocidad c_x Original.

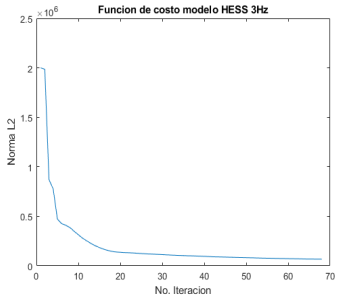
El valor de PSNR calculado para cada uno de las velocidades fueron de $34{,}2dB$, $36{,}6dB$ y $38{,}9dB$ para las velocidades c_x, d_x y c_z respectivamente.

Las funciones de costo para multifrecuencia se pueden ver en la figura 40. De manera general se observa un descenso en el error bastante bueno. Para las frecuencias de 3 y $9\ Hz$ requieren de entre 10 a 12 iteraciones para lograr un error aceptable, mientras que la inversión hecha a $6\ Hz$ solo necesitó de 4 iteraciones para llegar a un error bastante bajo, donde se truncó.

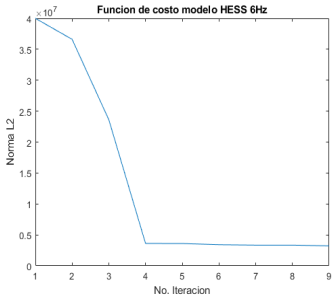
El modelo del Hess pese a ser el mas complejo a nivel físico tuvo un buen comportamiento a través de esta estrategia. Partiendo del modelo inicial, el resultado de la

inversión se asemeja mucho al modelo sintético. Algunas zonas conflictivas resultan de la zona de alta energía, pese a ello esas zonas no resultan tan afectadas. En general la inversión es acertada respecto a al modelo inicial y en gran medida resulta una mejor opción para hacer una inversión en comparación a los dos modelos anteriores, esto puede ser debido a que los parámetros tienen mayor sentido físico.

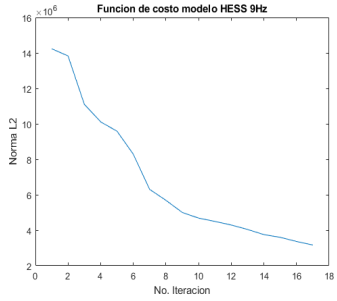
Figura 40: Función de costo para el modelo Hess con multifrecuencia en *hidden latency*



(a) Función de costo a 3 Hz para el modelo Hess.



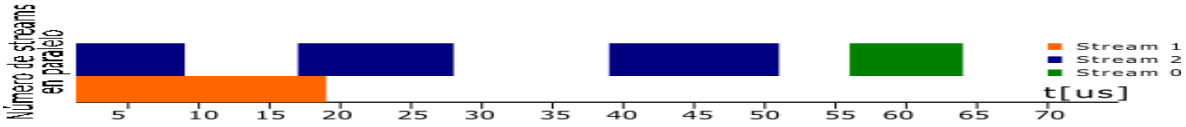
(b) Función de costo a 6 Hz para el modelo Hess.



(c) Función de costo a 9 Hz para el modelo Hess.

Las líneas de tiempo dadas en las figuras 41 y 42 representan la carga computacional de cada *kernel* tomando como referencia los tiempos de ejecución. También son una buena referencia para ver de mejor forma la manera en que se solapan las trasferencias de los datos junto con la ejecución de un *kernel*. Esto es sumamente ventajoso puesto que no se desperdicia tiempo durante las transferencias de datos y esto permite ver al *host* como una extensión de la RAM de la GPU.

Figura 41: Linea de tiempo para la propagación de la estrategia *hidden latency*.



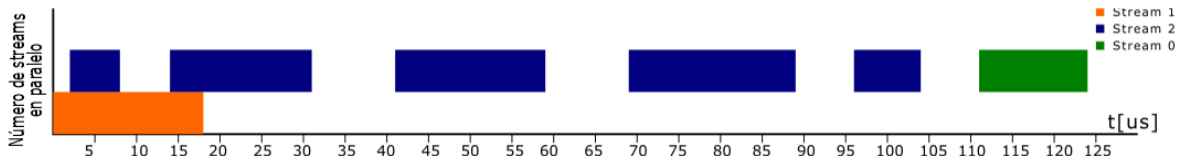
Cuadro 8: *Kernel* y tiempos de ejecución para la propagación en *hidden latency*.

Proceso de Propagación hacia adelante			
Kernels	Inicio de Ejecución[us]	Final de Ejecución[us]	Duración de Ejecución[us]
MemCpy[DtH]	0	19,008	19,008
Segunda Derivada	0	9,728	9,728
Psi	18	28,048	10,048
Zeta	39	50,265	11,265
Propagación	56	63,262	7,262

El cuadro 8 describe los *kernels* usados en la implementación de la propagación de esta estrategia, da detalles sobre los tiempos de inicio, finalización y ejecución de cada *kernel*. A diferencia de la estrategia del *checkpointing*, el *kernel* con más carga es el encargado de hacer las copias de información al *host*.

El cuadro 9 da detalles sobre los tiempos que toma cada *kernel* usado en la retropropagación para la estrategia. La carga computacional se ve más balanceada, gracias a que para la retropropagación dicha carga aumenta. Las copias de información siguen siendo el *kernel* de más duración a pesar de que los tiempos para los demás *kernels* ha aumentado.

Figura 42: Linea de tiempo para la retropropagación con la estrategia *hidden latency*.



Cuadro 9: *Kernels* y tiempos de ejecución para la retropropagación en *hidden latency*.

Proceso de Propagación Hacia Atrás			
	Inicio de Ejecución[us]	Final de Ejecución[us]	Duración de Ejecución[us]
MemCpy[HtD]	0	18,304	18,304
Punto de Entrada	2	8,624	6,624
Segunda Derivada	14	31,704	17,504
Psi	41	58,728	17,728
Zeta	69	89,480	20,48
RetroPropagación	96	103,712	7,712
Gradiente	111	123,630	12,640

Por otro lado, hay una desventaja en la estrategia, debido a lo múltiples flujos de ejecución, aparecen retardos entre el lanzamiento de los *kernels* de acuerdo al *stream* en que se hayan declarado como se explica en la sección 5.2. Esta estrategia respeta la línea de ejecución del algoritmo (Basados en las estrategias de operar solamente en la GPU). De esta manera es posible decir que no se hacen pasos de más, particularidad que la diferencia del *checkpointing*. Por otro lado, se generan ciertos retardos producto los múltiples contextos de ejecución que se crean. Estos retardos hacen que este proceso sea mas lento respecto a la estrategia tradicional.

El método de *hidden latency* aumenta en gran medida la cantidad de memoria a disposición de la GPU, por tanto, es posible operar modelos mucho más grandes puesto que se tiene a disposición la memoria del *host*.

En los cuadros 10 y 11 se puede ver la cantidad de llamados que se hace a cada uno de los *kernels* y las transferencias de datos hechas.
El cuadro 11, permite ver cómo las transferencias asíncronas agregan solamente un $0{,}99\,\%$ de carga temporal al sistema. Con respecto a la cantidad de memoria que reducen, que en total son dos volúmenes de $nt * nx * nz * 4$ bytes. El pago en tiempo es realmente pequeño considerando la carga mínima que agrega este tipo de transferencias.

Cuadro 10: Carga Computacional para la estrategia *hidden latency*.

Distribución de Carga Computacional		
Operación	Tiempo[%]	No. Llamadas
CudaMemCpy[DtH]	15.30 %	5005
CudaMemCpy[HtD]	14.66 %	5007
Propagado	14.07 %	10000
SegundaDerivada_P	11.71 %	10000
SegundaDerivada_BP	10.35 %	5000
Gradiente	7.90 %	5000
Propagador_HaciaAtrás	6.99 %	5000
PuntoDeEntrada	4.43 %	5000
Zeta_P	4.38 %	10000
Psi_P	3.83 %	10000
Zeta_BP	3.53 %	5000
Psi_Bp	2.86 %	5000

Cuadro 11: Operaciones realizadas por la estrategia *hidden latency*.

Operaciones del Sistema		
Operación	Tiempo[%]	No. Llamadas
kernels	73.15 %	70008
CudaMemSet	19.39 %	103
CudaMemCpy	3.52 %	12
Llamadas al Sistema	2.8 %	8013
CudaMemCpy[Async]	0.99 %	10000
CudaMalloc	0.15 %	46

Si se observa el cuadro 10 se puede ver en total cual es el aporte temporal de cada paso de la propagación y retropropagación en cuanto a llamadas al *toolkit* de Cuda. En este punto la carga computacional se obtiene asumiendo solo un contexto de ejecución por tanto no existen las transferencias asíncronas.

Durante las inversiones se tomaron los datos en cuanto a memoria ocupada y tiempo de ejecución. Los datos de memoria ocupada y tiempos de ejecución tomados durante la ejecución del algoritmo con la estrategia *hidden latency* para los modelos

del cuadrado difractor, logo de CPS y Hess están dados en los cuadros 12, 13 y 14 respectivamente.

Cuadro 12: Memoria y tiempo de ejecución para el modelo del cuadrado difractor con la estrategia *hidden latency*

Memoria en la GPU	Tiempo de ejecución del algoritmo
103[Mb]	$58,34$[s]

Cuadro 13: Memoria y tiempo de ejecución para el modelo del logo de CPS con la estrategia *hidden latency*

Memoria en la GPU	Tiempo de ejecución del algoritmo
102[Mb]	$56,92$[s]

Cuadro 14: Memoria y tiempo de ejecución para el modelo de Hess con la estrategia *hidden latency*

Memoria en la GPU	Tiempo de ejecución del algoritmo
478[Mb]	$105,42$[s]

Estos datos no son comparables con respecto al método de *checkpointing* en cuanto a tiempos de ejecución. Pero es posible compararlos con la memoria utilizada por la estrategia tradicional hablando específicamente del uso de la GPU. Se presentan los datos de memoria ocupada y tiempo de ejecución para los modelos del cuadrado difractor, el modelo de CPS y el modelo Hess en los cuadros 15, 16 y 17 respectivamente.

Cuadro 15: Memoria y tiempo de ejecución para el modelo del cuadrado con la estrategia tradicional.

Memoria en la GPU	Tiempo de ejecución del algoritmo
335[Mb]	$53,62$[s]

Cuadro 16: Memoria y tiempo de ejecución para el modelo del logo CPS con la estrategia tradicional.

Memoria en la GPU	Tiempo de ejecución del algoritmo
330[Mb]	$48,25$[s]

Cuadro 17: Memoria y tiempo de ejecución para el modelo Hess con la estrategia tradicional.

Memoria en la GPU	Tiempo de ejecución del algoritmo
4086[Mb]	$101,84$[s]

Haciendo la comparación con la estrategia tradicional, se pueden obtener porcentajes de aumento o disminución de uso de memoria en la GPU y de tiempos de ejecución para ambos métodos.
Para el método de *hidden latency*.

Cantidad de memoria usada en *hidden latency* con respecto a la estrategia tradicional:

1. Para el modelo del cuadrado se tiene un $69,25\,\%$ menos de memoria usada.
2. Con el modelo del logo de CPS se reducen en un $69,09\,\%$ la memoria.
3. Para el modelo de Hess, la reducción de memoria ocupada es del $88,30\,\%$.

En cuanto a tiempos de ejecución del algoritmo con *hidden latency* con respecto a la estrategia uno se tiene:

1. Para el modelo del cuadrado el algoritmo es un $8,09\,\%$ más lento.
2. Con el modelo del logo de CPS el aumento en tiempo de ejecución es del $15,23\,\%$.

3. Con el modelo Hess el aumento en tiempo es de tan sólo el $4,02\,\%$.

Para la estrategia del *checkpointing*.

Cantidad de memoria usada en *checkpointing* con respecto a la estrategia tradicional:

1. Para el modelo del cuadrado se tiene un $48,06\,\%$ menos de memoria usada.
2. Con el modelo del logo de CPS se reducen en un $48,18\,\%$ la memoria.
3. Para el modelo de Hess, la reducción de memoria ocupada es del $33,77\,\%$.

En cuanto a tiempos de ejecución del algoritmo con *checkpointing* con respecto a la estrategia uno:

1. Para el modelo del cuadrado el algoritmo es un $27{,}34\,\%$ más lento.
2. Con el modelo del logo de CPS tiempo de ejecución se aumenta en $43,68\,\%$.
3. Con el modelo Hess el aumento en tiempo es de $65,24\,\%$.

Aunque se espera un aumento en el tiempo de ejecución de los algoritmos con las estrategias implementadas, para el caso de la estrategia del *checkpointing* se tiene un aumento mayor del tiempo de ejecución. Este aumento puede ser causado por la propagación de más que se debe hacer para el cálculo del gradiente.

Cuadro 18: Parámetros y valores del experimento para el modelo cuadrado difractor

Parámetro	Descripción	*Hidden Latency*	*CheckPointing*
dh	Paso espacial	20m	20m
D	Tamaño del modelo	241 x 80	241 x 80
dt	Paso temporal	2e-3s	2e-3s
Tend	Tiempo de propagación	3.2s	3.2s
Si	Número de fuentes	12	12
Rec	Número de receptores	201	201
Vmax	Velocidad máxima	2000	2000
Vmin	Velocidad mínima	1500	1500
K	Número de iteraciones	172	170
a=cte	Número de iteraciones	6	6
L-BFGS	Número de iteraciones	166	164
Fsteps	Pasos de frecuencia	3, 6, 9 Hz	3, 6, 9 Hz
PSNR	Relación señal a ruido para c_x, d_x y c_z	36.35dB, 32.98dB y 34.77dB	35.46dB, 32.04dB y 34.12dB
RAM	Costo computacional	103Mb	161Mb
Time	Costo computacional	58.34s	68.36s

Cuadro 19: Parámetros del experimento para el modelo logo de CPS

Parámetro	Descripción	*Hidden Latency*	*Check Pointing*
dh	Paso espacial	20m	20m
D	Tamaño del modelo	221 x 87	221 x 87
dt	Paso temporal	2e-3s	2e-3s
Tend	Tiempo de propagación	3.2s	3.2s
Si	Número de fuentes	12	12
Rec	Número de receptores	181	221
Vmax	Velocidad máxima	1800	1800
Vmin	Velocidad mínima	1500	1500
K	Número de iteraciones	69	69
a=cte	Número de iteraciones	6	6
L-BFGS	Número de iteraciones	63	63
Fsteps	Pasos de frecuencia	3, 6, 9 Hz	3, 6, 9 Hz
PSNR	Relación señal a ruido para c_x, d_x y c_z	24.52dB, 19.56dB y 20.95dB	23.92dB, 18.86dB y 20.95dB
RAM	Costo computacional	103Mb	159Mb
Time	Costo computacional	56.92s	69.32s

Cuadro 20: Parámetros del experimento para el modelo Hess

Parámetro	Descripción	*Hidden Latency*	*Check Pointing*
dh	Paso espacial	20m	20m
D	Tamaño del modelo	841 x 270	841 x 270
dt	Paso temporal	2e-3s	2e-3s
Tend	Tiempo de propagación	4.2s	4.2s
Si	Número de fuentes	42	42
Rec	Número de receptores	801	801
Vmax	Velocidad máxima	4.3480e+03	4.3480e+03
Vmin	Velocidad mínima	1.4689e+03	1.4689e+03
K	Número de iteraciones	94	93
a=cte	Número de iteraciones	6	6
L-BFGS	Número de iteraciones	88	87
Fsteps	Pasos de frecuencia	3, 6, 9 Hz	3, 6, 9 Hz
PSNR	Relación señal a ruido para c_x, d_x y c_z	34.23dB, 36.57dB y 38.96dB	33.28dB, 35.40dB y 36.12dB
RAM	Costo computacional	478Mb	1380Mb
Time	Costo computacional	105.42s	168.32s

Se da una descripción de los parámetros que se utilizaron para la validación de las estrategias en los cuadros 18, 19 y 20 para los modelos del cuadrado difractor, logo de CPS y Hess respectivamente. En los cuadros se presentan los valores obtenidos para cada una de las estrategias implementadas.

7. CONCLUSIONES

En este trabajo se logró hacer la implementación del algoritmo de inversión de onda completa 2D para medios con anisotropía VTI utilizando estrategias computacionales para un mejor manejo de recursos de la GPU (memoria ocupada por el algoritmo y tiempo de ejecución).

En el capítulo 2 se da una descripción teórica de la técnica de inversión de onda completa para el caso de anisotropía VTI. Detalles de la implementación del algoritmo son tratados en el capítulo 3. Dicha implementación tiene ciertas dificultades que hacen necesaria la creación y aplicación de estrategias que hagan al algoritmo menos costoso computacionalmente hablando, dichos problemas son expuestos en el capítulo 4 y las estrategias desarrolladas e implementadas en el capítulo 5. Los resultados y el análisis de los mismos se presenta en el capítulo 6.

Con base en el desarrollo del trabajo descrito anteriormente, se puede concluir:

- La implementación de la FWI demanda una alta cantidad de recursos computacionales. Es bien sabido que utilizar arquitecturas secuenciales para implementar esta inversión resulta poco práctico debido a la cantidad de operaciones necesarias. Por esto, resulta necesario plantear estrategias computacionales para disminuir la cantidad de RAM usada y tiempos de ejecución. Las estrategias planteadas en este proyecto fueron dos, el método de *hidden latency* y el método del *checkpointing*.

  Con base en los resultados que se obtuvieron es posible afirmar que hay algunas desventajas en ambos métodos. Para el caso del método del *checkpointing*, las desventajas que presenta son el error asociado al truncamiento de datos debido a las propagaciones parciales que se deben hacer durante la aplicación del método y la propagación extra que se debe hacer para el cálculo del gradiente. Para el método de *hidden latency*, la desventaja es el aumento en tiempo de ejecución debido a los diferentes contextos de ejecución que se trabajan en el método.

- Las estrategias implementadas cumplieron con el objetivo por el que fueron utilizadas, se logró una reducción sustancial en la memoria usada por el algoritmo en comparación con la memoria ocupada por la implementación tradicional. Para el método del *checkpointing*, la reducción de memoria usada es de aproximadamente un $44\,\%$, promediando la reducción de memoria para cada modelo.
  Para la estrategia de *hidden latency* la reducción en RAM usada, promediando entre el uso de memoria para cada modelo, es del $75\,\%$ aproximadamente. Se demuestra que las estrategias cumplen con el objetivo de disminuir la RAM de la GPU que se utiliza para el algoritmo.
  Resulta conveniente hacer comparaciones entre estrategias de a cuerdo al error o integridad de los datos manejados por la FWI. De acuerdo con lo anterior, la estrategia *hidden latency* tiene un mejor manejo de datos pues no tiene truncamiento de datos como la estrategia del *checkpointing*. *Hidden latency* garantiza que las copias de información hechas entre el *host* y la GPU no se corrompan, asegura la integridad de los datos.

- El método de *checkpointing*, tiene que hacer truncamiento y aproximación de datos debido a las propagaciones y retropropagaciones parciales que se deben hacer dentro de su desarrollo. Esto introduce un error que afecta al cálculo del gradiente, haciendo que la inversión no sea tan precisa como debería. Esto se puede apreciar en los valores de PSNR obtenidos para cada modelo.
  Los valores de PSNR arrojados por la estrategia *hidden latency* tiene en promedio un valor de $34{,}7dB$ para el modelo del cuadrado difractor, $21{,}7dB$ para el logo de CPS y $36{,}6dB$ frente a $33{,}8dB$, $21,24dB$ y $34{,}9dB$ para los modelos del cuadrado, CPS y Hess respectivamente que tiene la estrategia de *chekpointing*. *Hidden latency* tiene siempre mejores valores de PSNR que *checkpointing*.

- Ambas estrategias implementadas fueron sometidas a pruebas con diferentes tamaños de modelo para observar el rendimiento que tienen al manejar grandes cantidades de datos. El modelo más grande que se usó para validar las estrategias fue el modelo Hess que tiene un tamaño de 841x270 puntos. De acuerdo a tiempos de ejecución, memoria ahorrada y PSNR, se puede argumentar cual es-

trategia es mejor para grandes cantidades de datos. La estrategia *checkpointing* tuvo un aumento del $39\,\%$ en tiempo de ejecución, se ahorró un $33\,\%$ de memoria y se obtuvo un valor promedio de PSNR de $-114dB$. La estrategia de *hidden latency* tuvo un aumento del $4\,\%$ en tiempo de ejecución, redujo en un $88\,\%$ la memoria usada y el modelo final tiene en promedio un PSNR de $-106dB$. Por lo anterior se puede decir que la estrategia del *hidden latency* es la mejor opción para hacer inversiones de modelos grandes. Las características de concurrencia que tiene la estrategia hacen que se tenga un mejor desempeño con grandes cantidades de datos.

8. RECOMENDACIONES

- Al usar MPI la cantidad de memoria reservada en el *host* depende de la cantidad de nodos que se ejecuten. Por lo tanto es recomendable, para trabajos futuros, hacer reservas de memoria por nodo. De esa manera se asegura tener una memoria maestra la cual es compartida por todos los nodos y se tiene también memoria independiente para cada nodo. Se recomienda una buena gestión de la memoria disponible en el clúster para disminuir carga computacional, tanto en memoria como tiempo de ejecución del algoritmo en su totalidad.

- Se recomienda reducir la cantidad de cálculos se deben hacer en la propagación y retropropagación. Para esto es aconsejable limitar el cálculo de las variables auxiliares psi y $zeta$ de la zona CPML para que operen únicamente en dicha región CPML. Evitar el cálculo de estas variables sobre el modelo total se puede traducir en la reducción de hasta aproximadamente un $20\,\%$ en el tiempo de ejecución de la propagación y retropropagación.

- Si se quisiera dar continuidad a este trabajo, se sugiere jugar con los parámetros de L-BFGS ya que se puede encontrar una combinación de parámetros que puedan enfilar a mejores resultados de inversión. Ya que el método del descenso del gradiente maneja un α constante que puede indicar cuando se llega a un mínimo local. Se puede pensar que haciendo una implementación que intercale el uso del gradiente descendiente y L-BFGS se pueda lograr un mejor descenso en la función de costo.

REFERENCIAS

[1] ABREO David, “Evaluación de estrategias de implementación de una inversión de onda completa (fwi) 3d acústica con densidad constante en el dominio temporal sobre una arquitectura gpu.,” Master’s thesis, Universidad Industrial de Santantader, 2016.

[2] ABREO CARRILLO Sergio Alberto *et al.*, “A practical implementation of acoustic full waveform inversion on graphical processing units,” *CT&F - Ciencia, Tecnología y Futuro*, vol. 6, pp. 5 – 16, 07 2015.

[3] ANDERSON John, TAN Lijian, and WANG Don, “Time-reversal checkpointing methods for rtm and fwi,” *GEOPHYSICS*, vol. 77, no. 4, pp. S93–S103, 2012.

[4] BOONYASIRIWAT Chaiwoot *et al.*, “An efficient multiscale method for time-domain waveform tomography,” *GEOPHYSICS*, vol. 74, no. 6, pp. WCC59–WCC68, 2009.

[5] BRITTAN John *et al.*, “Full waveform inversion - the state of the art,” vol. 31, pp. 75–81, 10 2013.

[6] COLLINO Francis and TSOGKA Chrysoula, “Application of the perfectly matched absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media,” *GEOPHYSICS*, vol. 66, no. 1, pp. 294–307, 2001.

[7] Glossary Schlumberger Oilfield, “Nmo.” On-line, Available in: `http://www.glossary.oilfield.slb.com/en/Terms/n/nmo.aspx`, Accessed: 04-2018.

[8] GROPP William *et al.*, *Using Advanced MPI: Modern Features of the Message-Passing Interface*, MIT Press, 2014.

[9] GÓMEZ Douglas, CENTENO Mario, and CHÁVEZ Sergio, “Anisotropía sismica: Una breve revisión general,” tech. rep., Instituto Mexicano de Petróleos, 2015.

[10] HARRIS Mark, “How to optimize data transfers in cuda c/c++,” tech. rep., https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/, 2012.

[11] HICKS Graham and PRATT Gerhard, “Reflection waveform inversion using local descent methods: Estimating attenuation and velocity over a gas-sand deposit,” *GEOPHYSICS*, vol. 66, no. 2, pp. 598–612, 2001.

[12] JONES Ian, *Incorporating Near-Surface Velocity Anomalies in Pre-Stack Depth Migration Models*, pp. 1–1, 2015.

[13] MA Yong and HALE Dave, “Quasi-newton full-waveform inversion with a projected hessian matrix,” *GEOPHYSICS*, vol. 77, no. 5, pp. R207–R216, 2012.

[14] MARTINS Marcio *et al.*, “Modelagem e migração sísmica 2d em meios transversalmente anisotrópicos-vti,” tech. rep., 2010.

[15] NOCEDAL Jorge and WRIGHT Stephen, *Numerical Optimization*, New York, NY, USA: Springer, second ed., 2006.

[16] NVIDIA, “Gpu-based computing,” tech. rep., http://slideplayer.com/slide/8715304/.

[17] NVIDIA, “What is gpu-accelerated computing?,” tech. rep., http://www.nvidia.com/object/what-is-gpu-computing.html.

[18] OPEN_MPI_TEAM, “Mpi,” tech. rep., https://www.open-mpi.org/doc/current/man3/MPI.3.php#sect1.

[19] OPERTO Stéphane *et al.*, “Quantitative imaging of complex structures from dense wide-aperture seismic data by multiscale traveltime and waveform inversions: A case study,” vol. 52, pp. 625–651, 11 2004.

[20] PASALIC Damir and MCGARRY Ray, *Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations*, pp. 2925–2929, 2010.

[21] PLESSIX Rene-Edouard, “A review of the adjoint-state method for computing the gradient of a functional with geophysical applications,” *Geophysical Journal International*, vol. 167, no. 2, pp. 495–503, 2006.

[22] PÉREZ Carlos Andrés, *Two-dimensional near-surface seismic imaging with surface waves : alternative methodology for waveform inversion*. Theses, Ecole Nationale Supérieure des Mines de Paris, Dec. 2013.

[23] RENNICH Steve, "Cuda c/c++ streams and concurrency," tech. rep., NVIDIA.

[24] SANTOS Adriano DOS and PESTANA Reynam, "Time-domain multiscale full-waveform inversion using the rapid expansion method and efficient step-length estimation," *GEOPHYSICS*, vol. 80, no. 4, pp. R203–R216, 2015.

[25] SAVA Paul and BIONDI Biondo, "Wave-equation migration velocity analysis. i. theory," *Geophysical Prospecting*, vol. 52, no. 6, pp. 593–606, 2004.

[26] THOMSEN Leon, "Weak elastic anisotropy," *GEOPHYSICS*, vol. 51, no. 10, pp. 1954–1966, 1986.

[27] TORRES Jairo, "Modelado acústico y migración pspi en medios con anisotropía polar," Master's thesis, Universidad EAFIT, 2010.

[28] VIRIEUX Jean and OPERTO Stéphane, "An overview of full-waveform inversion in exploration geophysics," *GEOPHYSICS*, vol. 74, no. 6, pp. WCC1–WCC26, 2009.

[29] WANG Chao *et al.*, *VTI Waveform Inversion with Practical Strategies: Application to 3D Real Data*, pp. 1–6, 2012.

[30] WANG Hui and SAVA Paul, "Pseudo-acoustic wavefield tomography with model constraints,", 2015.

[31] WARNER Michael *et al.*, "Anisotropic 3d full-waveform inversion," *GEOPHYSICS*, vol. 78, no. 2, pp. R59–R80, 2013.

[32] WESDORP Jaap J., "Method for the adjoint system and model parameter gradients.", Grupo de investigación CPS, unpublished 2018.

[33] ZHANG Meng *et al.*, "Full waveform inversion on the cpu/gpu heterogeneous platform and its application analysis," vol. 53, pp. 461–467, 07 2014.

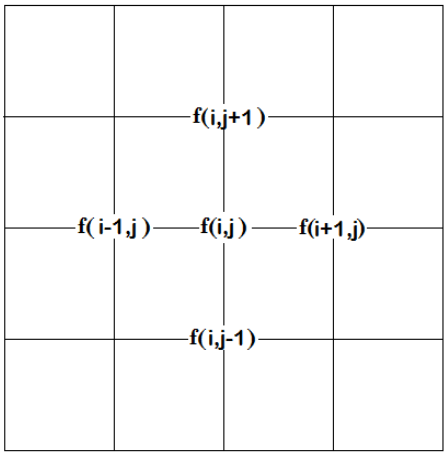
ANEXOS

A. Discretización de las ecuaciones de onda

Para la implementación del algoritmo, se usa el set de ecuaciones dadas en 3. Este procedimiento de discretización se hace usando la técnica de diferencias finitas de octavo orden en espacio y segundo orden en tiempo.

La discretización se hace con diferencias finitas centradas, es decir la aproximación de una derivada se hace agregando igual cantidad de términos tanto adelante como atrás en espacio tanto para movimiento horizontal como movimiento vertical. Esto se explica en el diagrama 12 y se puede ver mejor en la figura .1.

Figura .1: Posiciones en una malla para la discretización.



Siguiendo lo anterior, la discretización de las ecuaciones de onda 3 se da como

$$\frac{P_{i,j}^{n+1}-2P_{i,j}^{n}+P_{i,j}^{n-1}}{\Delta t^2}=\frac{c_x}{\Delta x^2}[-\frac{1}{560}\left(P_{i-4,j}+P_{i+4,j}\right)+\frac{8}{315}\left(P_{i-3,j}+P_{i+3,j}\right)$$
$$-\frac{1}{5}\left(P_{i-2,j}+P_{i+2,j}\right)+\frac{8}{5}\left(P_{i-1,j}+P_{i+1,j}\right)-\frac{205}{72}P_{i,j}]+\frac{c_z}{\Delta z^2}[-\frac{1}{560}\left(R_{i,j-4}+R_{i,j+4}\right)$$
$$+\frac{8}{315}\left(R_{i,j-3}+R_{i,j+3}\right)-\frac{1}{5}\left(R_{i,j-2}+R_{i,j+2}\right)+\frac{8}{5}\left(R_{i,j-1}+R_{i,j+1}\right)-\frac{205}{72}R_{i,j}],$$

$$\frac{R_{i,j}^{n+1}-2R_{i,j}^{n}+R_{i,j}^{n-1}}{\Delta t^2}=\frac{d_x}{\Delta x^2}[-\frac{1}{560}\left(P_{i-4,j}+P_{i+4,j}\right)+\frac{8}{315}\left(P_{i-3,j}+P_{i+3,j}\right)$$
$$-\frac{1}{5}\left(P_{i-2,j}+P_{i+2,j}\right)+\frac{8}{5}\left(P_{i-1,j}+P_{i+1,j}\right)-\frac{205}{72}P_{i,j}]+\frac{d_z}{\Delta z^2}[-\frac{1}{560}\left(R_{i,j-4}+R_{i,j+4}\right)$$
$$+\frac{8}{315}\left(R_{i,j-3}+R_{i,j+3}\right)-\frac{1}{5}\left(R_{i,j-2}+R_{i,j+2}\right)+\frac{8}{5}\left(R_{i,j+1}+R_{i,j-1}\right)-\frac{205}{72}R_{i,j}]. \tag{59}$$

Los coeficientes que acompañan a los términos de las ondas P y R están dados por la técnica de las diferencias finitas. Una vez discretizadas las ecuaciones, se pueden ordenar, agrupándolas por coeficientes semejantes.
Para la implementación estas ecuaciones se seccionan y se guardan en variables temporales como se muestra en las ecuaciones 13 y 14.

Para la discretización de la ecuación 4 se toma la misma plantilla anterior haciendo la discretización en octavo orden espacial y segundo orden temporal como se muestra

$$\frac{P_{i,j}^{n+1}-2P_{i,j}^{n}+P_{i,j}^{n-1}}{\Delta t^2}=\frac{c_x}{\Delta x^2}[-\frac{1}{560}\left(P_{i-4,j}+P_{i+4,j}\right)+\frac{8}{315}\left(P_{i-3,j}+P_{i+3,j}\right)$$
$$-\frac{1}{5}\left(P_{i-2,j}+P_{i+2,j}\right)+\frac{8}{5}\left(P_{i-1,j}+P_{i+1,j}\right)-\frac{205}{72}P_{i,j}]+\frac{c_z}{\Delta z^2}[-\frac{1}{560}\left(R_{i,j-4}+R_{i,j+4}\right)$$
$$+\frac{8}{315}\left(R_{i,j-3}+R_{i,j+3}\right)-\frac{1}{5}\left(R_{i,j-2}+R_{i,j+2}\right)+\frac{8}{5}\left(R_{i,j-1}+R_{i,j+1}\right)-\frac{205}{72}R_{i,j}]$$
$$+c_x[\frac{1}{\Delta h}(\frac{1}{280}\left(\psi_{i-4,j}-\psi_{i+4,j}\right)+\frac{4}{105}\left(\psi_{i+3,j}-\psi_{i-3,j}\right)+\frac{1}{5}\left(\psi_{i-2,j}-\psi_{i+2,j}\right)$$
$$+\frac{4}{5}\left(\psi_{i+1,j}-\psi_{i-1,j}\right))+\zeta_{i,j}]+c_z[\frac{1}{\Delta h}(\frac{1}{280}\left(\psi_{i,j-4}-\psi_{i,j+4}\right)+\frac{4}{105}\left(\psi_{i,j+3}-\psi_{i,j-3}\right)$$
$$+\frac{1}{5}\left(\psi_{i,j-2}-\psi_{i,j+2}\right)+\frac{4}{5}\left(\psi_{i,j+1}-\psi_{i,j-1}\right))+\zeta_{i,j}],$$

$$\frac{R_{i,j}^{n+1}-2R_{i,j}^{n}+R_{i,j}^{n-1}}{\Delta t^2}=\frac{d_x}{\Delta x^2}[-\frac{1}{560}\left(P_{i-4,j}+P_{i+4,j}\right)+\frac{8}{315}\left(P_{i-3,j}+P_{i+3,j}\right)$$
$$-\frac{1}{5}\left(P_{i-2,j}+P_{i+2,j}\right)+\frac{8}{5}\left(P_{i-1,j}+P_{i+1,j}\right)-\frac{205}{72}P_{i,j}]+\frac{d_z}{\Delta z^2}[-\frac{1}{560}\left(R_{i,j-4}+R_{i,j+4}\right)$$
$$+\frac{8}{315}\left(R_{i,j-3}+R_{i,j+3}\right)-\frac{1}{5}\left(R_{i,j-2}+R_{i,j+2}\right)+\frac{8}{5}\left(R_{i,j-1}+R_{i,j+1}\right)-\frac{205}{72}R_{i,j}]$$
$$+c_x[\frac{1}{\Delta h}(\frac{1}{280}\left(\psi_{i-4,j}-\psi_{i+4,j}\right)+\frac{4}{105}\left(\psi_{i+3,j}-\psi_{i-3,j}\right)+\frac{1}{5}\left(\psi_{i-2,j}-\psi_{i+2,j}\right)$$
$$+\frac{4}{5}\left(\psi_{i+1,j}-\psi_{i-1,j}\right))+\zeta_{i,j}]+c_z[\frac{1}{\Delta h}(\frac{1}{280}\left(\psi_{i,j-4}-\psi_{i,j+4}\right)+\frac{4}{105}\left(\psi_{i,j+3}-\psi_{i,j-3}\right)$$
$$+\frac{1}{5}\left(\psi_{i,j-2}-\psi_{i,j+2}\right)+\frac{4}{5}\left(\psi_{i,j+1}-\psi_{i,j-1}\right))+\zeta_{i,j}]. \tag{60}$$

Estas ecuaciones también se pueden ordenar y seccionar para facilitar la implementación. El resultado de lo anterior se muestra en las ecuaciones 15, 16, 17 y 18.