Estrategia de implementación de la Inversión de Onda Completa (FWI) 2D Elástica en el dominio del tiempo en GPU

Anderson Páez Chanagá

Tesis de investigación presentada como requerimiento para optar al título de

Magister en Ingeniería Electrónica

Director:

Ph.D Ana Beatriz Ramírez Silva

Co-director:

MSc. Iván Javier Sánchez Galvis

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2021

Dedicatoria

Este trabajo quiero dedicarlo primero a Dios, a mi madre Maria Edilia Chanagá, mi hermano Yovanny Cartagena y mi esposa Laura Judith Alvarado por su paciencia y apoyo incondicional durante todo el proceso vivido hasta la culminación de este logro personal y profesional.

Una mención especial a mi suegro Mario Alvarado, un luchador caído en esta guerra contra la pandemia.

A ellos toda mi admiración, cariño y respeto, este logro también es suyo. ✠

Agradecimientos

Quiero agradecer al grupo de investigación CPS por haber apoyado la candidatura a la maestría, especial reconocimiento a la profesora Ana Beatriz por confiar en mis capacidades y haberme dado la oportunidad de afrontar el reto de esta formación.

Al Co-op Iván Sánchez por la paciencia en sus explicaciones cuando lo requerí; igualmente por los *tinto times* junto a mi buen amigo Jheyston Omar a quien también agradezco su disposición de apoyo en momentos cuando me encontraba estancado en la investigación.

Quiero también agradecer a los profesores, personal administrativo y compañeros estudiantes E3T que me acompañaron durante el proceso pues con sus aportes me han ayudado a crecer un poco más como persona y como profesional.

Tabla de Contenido

Introducción **14**

1. Objetivos **17**

2. Modelado elástico 2D **18**

2.1. Formulación del modelado de onda elástico 19

2.2. Comportamiento de bordes reflectivos e implementación de CPML 21

2.3. Discretización por diferencias finitas (FD) 25

3. Inversión de onda completa (FWI) **31**

3.1. Ecuaciones adjuntas y retro propagación 35

3.2. Cálculo de los gradientes de parámetros elásticos 37

3.3. FWI es un problema *mal puesto* 39

4. Computación de alto desempeño (HPC) **41**

4.1. Computación paralela 42

4.2. Computación heterogénea 45

4.3. Unidades de procesamiento gráfico (GPU) 47

4.4. Nvidia CUDA C framework 48

5. Estrategia computacional para la implementación de FWI 2D elástica **54**

5.1. Consideraciones para el diseño de hilos en GPU para la FWI elástica 56

5.2. Implementación de CPML en GPU, estrategia común y de memoria reducida 59

5.3. Administración de memoria CPU-GPU para almacenamiento del cubo de propagación 62

5.4. Estrategia computacional para la implementación de FWI 2D elástica en GPU usando el paradigma heterogéneo 66

6. Resultados **72**

6.1. Modelo CPS 73

6.1.1. Medidas de ajuste para la FWI 73

6.1.2. Medición del desempeño computacional de la FWI 82

6.2. Modelo Overthrust 2D 88

6.2.1. Medidas de ajuste para la FWI 88

6.2.2. Medición del desempeño computacional de la FWI 95

7. Conclusiones **98**

7.1. Trabajo futuro 100

Bibliografía **102**

Apéndices **107**

Lista de Figuras

Figura 1. Capturas para la simulación del modelado de onda en una zona de estudio de 535 [m]x268 [m]. 21

Figura 2. Modelo donde se localizan las zonas regular y absorbente. 22

Figura 3. Capturas para la simulación del modelado de onda en una zona de estudio de 535 [m]x268 [m] con la aplicación del método CPML para evitar las reflexiones en los bordes. 26

Figura 4. Distribución de cuadrícula del modelo discretizado. 27

Figura 5. Malla intercalada en tiempo-espacio (3D) para la solución de los campos de velocidad y esfuerzo en las ecuaciones de propagación. 28

Figura 6. Representación usual 2D de la malla intercalada mostrando solo la distribución espacial considerando los campos de velocidad proyectados sobre los campos de esfuerzos. 29

Figura 7. Ejemplo de un sismograma real, tomado de Duarte et al. (2014). 31

Figura 8. Ejemplos de sismogramas sintéticos, a la izquierda, el sismograma verdadero, a la derecha, el sismograma modelado. 33

Figura 9. Malla intercalada tiempo-espacio (3D) para la solución de los campos adjuntos. 37

Figura 10. Función de costo hipotética y localización de mínimos locales y globales, el problema mal puesto sucede porque un mal modelo inicial puede llevar hacia un mínimo local. 40

Figura 11. Programación secuencial donde las instrucciones son ejecutadas una a una en línea, la última instrucción debe esperar a que las anteriores terminen. Tomado de Cheng et al. (2014). 43

Figura 12. Programación paralela donde múltiples instrucciones sencillas son ejecutadas de forma simultánea junto con las secuenciales. Tomado de Cheng et al. (2014). 44

Figura 13. Implementación del paradigma heterogégeno. Tomado de Cheng et al. (2014). 46

Figura 14. Arquitectura GPU Nvidia Turing. Tomado de NVIDIA (2018). 49

Figura 15. Esquema de diseño de hilos para implementar paralelismo en tarjetas GPU Nvidia. Tomado de Cheng et al. (2014). 52

Figura 16. Ejemplos de diseño de hilos para un kernel que se ejecuta sobre 16 hilos: a) diseño 1D-1D , b) diseño 1D-2D , c) diseño 1D-2D , d) diseño 2D-2D. 53

Figura 17. Diagrama de flujo general para la implementación de la FWI elástica en GPU. 54

Figura 18. Opciones de diseño de hilos para la implementación de FWI elástica: a) diseño 1D, b) diseño 2D. 57

Figura 19. Distribución de las zonas CPML asignación de recursos: a) estrategia común para la asignación de CPML, b) Estrategia de memoria reducida CPML. 60

Figura 20. Almacenamiento del cubo de propagación en la memoria de sistema (RAM) de la CPU cada vez que la GPU actualiza el campo. 64

Figura 21. Almacenamiento del cubo de propagación en la memoria global de la GPU (VRAM) cada vez que la GPU actualiza un campo. 65

Figura 22. Diagrama de flujo para la estrategia de implementación computacional de la FWI 2D elástica utilizando el paradigma heterogéneo. 68

Figura 23. Modelo sintético verdadero CPS para evaluar la FWI programada. 74

Figura 24. Modelos iniciales CPS para las pruebas de FWI, el modelo inicial homogéneo (izquierda), el modelo inicial filtrado gaussiano (derecha). 76

Figura 25. Resultados de la FWI partiendo del modelo inicial homogéneo (izquierda), modelo final después de 100 iteraciones (centro), modelo final después de 500 iteraciones (derecha). 78

Figura 26. Métricas para evaluar el ajuste de la FWI partiendo del modelo inicial homogéneo, MSE (izquierda), función de costo (derecha). 79

Figura 27. Resultados de la FWI partiendo del modelo inicial filtrado gaussiano (izquierda), modelo final después de 100 iteraciones (centro), modelo final después de 500 iteraciones (derecha). 80

Figura 28. Métricas para evaluar el ajuste de la FWI en el modelo CPS partiendo del modelo inicial filtrado gaussiano, MSE (izquierda), función de costo (derecha). 82

Figura 29. Modelos finales CPS resultado de implementación de FWI, partiendo de modelo inicial homogéneo (izquierda), partiendo de modelo inicial filtrado gaussiano (derecha). 83

Figura 30. Captura del Nvidia Visual Profiler recolectando información sobre el desempeño de la estrategia FWI implementada sobre el modelo CPS. 85

Figura 31. Captura de Nvidia nvtop para determinar la cantidad de memoria VRAM consumida por la implementación FWI ejecutando sobre el modelo CPS. 86

Figura 32. Modelo verdadero sintético Overthrust 2D para pruebas de FWI. 89

Figura 33. Comparación de la inversión para el modelo Overthrust 2D: modelo inicial filtrado gaussiano (izquierda), modelo final después de 800 iteraciones (derecha). 91

Figura 34. Resultado de la inversión en la zona de acercamiento 1 para la inversión del modelo Overthrust: modelo inicial filtrado gaussiano (izquierda), modelo final después de 800 iteraciones (centro), modelo verdadero (derecha). 92

Figura 35. Resultado de la inversión en la zona de acercamiento 2 para la inversión del modelo Overthrust: modelo inicial filtrado gaussian (izquierda), modelo final después de 800 iteraciones (centro), modelo verdadero (derecha). 93

Figura 36. Métricas para evaluar el ajuste de la FWI en el modelo Overthrust 2D partiendo del modelo inicial filtrado gaussiano, MSE (izquierda), función de costo (derecha). 94

Figura 37. Comparación entre el modelo final después de la FWI de 800 iteraciones (izquierda) y el modelo verdadero (derecha). 94

Figura 38. Captura del Nvidia Visual Profiler recolectando información sobre el desempeño de la estrategia FWI implementada sobre el modelo Overthrust 2D. 95

Figura 39. Captura de Nvidia nvtop para determinar la cantidad de memoria VRAM consumida por la implementación FWI ejecutando sobre el modelo Overthust 2D. 97

Lista de Apéndices

pág.

Apéndice A. Discretización por diferencias finitas (FD) de segundo orden en tiempo y cuarto en espacio de las ecuaciones de onda elásticas 107

Apéndice B. Discretización por diferencias finitas (FD) de segundo orden en tiempo y cuarto en espacio de las ecuaciones de campos adjuntos 109

Apéndice C. Pseudocódigo para la implementación de FWI elástica 111

Resumen

Título: Estrategia de implementación de la Inversión de Onda Completa (FWI) 2D Elástica en el dominio del tiempo en GPU [*]

Autor: Anderson Páez Chanagá [**]

Palabras Clave: Inversión de Onda Completa, Modelado de Onda Elástico, Computación de Alto Desempeño, GPU

Descripción: La inversión de onda completa (FWI) elástica 2D es un procedimiento utilizado en la industria de Oil & Gas para obtener imágenes del subsuelo de alta resolución con el fin de facilitar la localización de dichos minerales; sin embargo, su implementación presenta un reto computacional debido a la enorme cantidad de información por procesar y los amplios tiempos de ejecución. En este trabajo fue desarrollada una estrategia computacional para implementar FWI usando técnicas HPC que incluyen paralelismo en GPU y programación secuencial en CPU. La estrategia explora 3 tácticas que involucran el diseño de hilos 2D, la memoria reducida en CPML y almacenamiento de campo en VRAM; de esta forma se busca reducir la cantidad de memoria utilizada y usar más eficientemente los recursos de la GPU. La estrategia fue evaluada usando 2 modelos (CPS y Overthrust 2D) con distintos niveles de complejidad, posteriormente se evalúan los modelos finales de parámetros elásticos por inspección visual y usando las métricas MSE y función de costo. El desempeño computacional fue medido a través del tiempo de ejecución, porcentaje de utilización de GPU y consumo de VRAM. Los resultados muestran que el algoritmo con la estrategia propuesta es funcional generando imágenes de reconstrucción de los parámetros adecuados incluso al considerar modelos complejos, estos resultados sugieren que la implementación de FWI 2D elástica programada puede ser utilizada con modelos reales.

---

[*] Tesis de Maestria

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Ana Beatriz Ramírez Silva, Doctorado en Ingeniería Eléctrica. Codirector: Iván Javier Sánchez Galvis, Magister en Ingeniería Electrónica.

Abstract

Title: Strategy for implementation of elastic 2D Full Waveform Inversion (FWI) in time using a GPU [*]

Author: Anderson Páez Chanagá [**]

Keywords: Full Waveform Inversion, Elastic Wave Modeling, High Performance Computing. GPU

Description: Elastic 2D Full Waveform Inversion (FWI) is a state-of-art procedure used in Oil & Gas industry to obtain high resolution subsurface earth images that boost the mineral exploration procedures by easing the location of those resources; however, it presents a computational challenge to implement due to the enormous amount of data to process and extended execution times. In this work, it was developed a computational strategy for programming the FWI algorithm which take advantage of newly HPC computing by joining forces of parallelism in GPU and sequential programming in CPU. The strategy includes three computational techniques which are 2D thread layouts, CPML reduced memory and VRAM storage of wavefield propagation; in this way, it is intended to reduce the total memory consumed and to use the GPU resources efficiently. The strategy was evaluated by executing in 2 different models (CPS and Overthrust 2D) which have distinct levels of complexity and evaluating the final models for elastic parameters obtained both visual and using MSE- cost function adjustment metrics. Computational performance was measured through execution time, compute utilization and VRAM consumption metrics. Results show that the algorithm with proposed strategy is fully functional in tests giving accurate reconstructed images of elastic parameters considering both simple or complex models, those results permit to suggest that the elastic 2D FWI implementation could be tested with real models.

---

[*] Master Thesis

[**] Faculty of Physicomechanical Engineering. School of Electrical, Electronic and Telecomunications Engineering. Director: Ana Beatriz Ramírez Silva, Ph.D in Electrical Engineering. Co-director: Iván Javier Sánchez Galvis, MSc. in Electrical Engineering

Introducción

La extracción de minerales desde las profundidades de la tierra es una actividad compleja y costosa debido, entre otras razones, a la incertidumbre asociada con la localización de los recursos que se quieren explotar donde se incluyen los hidrocarburos, gas y otros.

Para tratar con el problema de la incertidumbre, el procesamiento de imágenes sísmicas busca generar imágenes (normalmente llamadas modelos) detallando la composición bajo tierra por medio de un análisis de información recopilada por las ondas sísmicas cuando se propagan en determinada área de estudio; esta información resulta útil en los procesos de toma de decisiones en proyectos de extracción; debido a esto el manejar herramientas que permitan obtener estos modelos en corto tiempo y a un costo reducido es una necesidad real de la industria.

Existe varios métodos para el procesamiento de imágenes sísmicas que generan los modelos mencionados, entre ellos son ampliamente reconocidos los métodos de migración e inversión, sin embargo, en los últimos años ha tomado bastante interés la Inversión de onda completa (FWI por sus siglas en inglés) debido a su capacidad para construir modelos del subsuelo de alta resolución. Sin embargo, la apropiada implementación computacional es bastante complicada de lograr debido a la enorme cantidad de información y el alto costo de computacional de procesamiento requerido para alcanzar resultados que sean útiles.

En la búsqueda de nuevas formas para solventar la dificultad de implementación, una propuesta consiste en trabajar sobre una versión mas simple de FWI como lo es aquella que utiliza la formulación acústica; esta versión no considera parte de la información recolectada por la onda en

el procesamiento de datos pero permite mejorar el tiempo computacional en ejecutar el método. A pesar de que esta versión de la FWI ha sido ampliamente estudiada e implementada en los últimos 20 años, la no consideración de información de la onda en el procesamiento puede llevar en ciertos aspectos a obtener modelos *incompletos* del subsuelo.

Debido a lo mencionado anteriormente, ha surgido la necesidad de lograr modelos mas realistas al considerar mayor cantidad de información contenida en la onda propagada, esto lleva a formulaciones mas complejas como la presentada en la versión elástica de FWI, sin embargo, lograr un implementación adecuada de esta versión ha sido difícil debido al incremento del costo computacional al utilizar la formulación elástica y por tanto este tipo de inversión se encuentra en investigación continua.

Intentar solucionar el problema de alta demanda computacional asociado con la implementación de FWI ha sido una de las grandes preocupaciones de los investigadores desde hace 30 años donde se contaba con estaciones de trabajo con capacidades de procesamiento y almacenamiento reducidas incapaces muchas veces de tratar con la carga computacional, sin embargo, el uso de las unidades de procesamiento gráfico (GPU por sus siglas en inglés) en tareas distintas a la ejecución de videojuegos o GPGPU, junto con el desarrollo de la computación de alto desempeño (HPC por su sigla en inglés) apoyado por la *computación heterogenea* que ha impulsado la *supercomputación*; ha permitido en la actualidad enfrentar este problema del alto costo computacional relacionado con la implementación de FWI.

En este trabajo, se presentará una estrategia para implementar la inversión de onda completa 2D elástica soportada en un sistema HPC que incluye una GPU para el procesamiento de datos, pa-

ra lograrlo, se utilizará el paradigma de programación heterogénea usando una estación de trabajo que incluye CPU y GPU implementando programación secuencial y paralela; el algoritmo de FWI será probado usando dos modelos sintéticos con diferentes niveles de complejidad y los resultados serán evaluados utilizando medidas de nivel de ajuste del modelo y de desempeño computacional del algoritmo implementado.

Esta tesis esta organizada como se presenta a continuación: el capítulo 2 contiene la formulación para implementar modelado 2D elástico y las consideraciones computacionales del método de discretización por diferencias finitas. En el capítulo 3, se presenta la teoría relacionada con el método de estado adjunto y la retro propagación, el cálculo de los gradientes y finalmente se presenta la inversión de onda completa y las entradas necesarias para poder ser ejecutada. En el capítulo 4, se introduce brevemente la computación HPC, la computación heterogénea, la arquitectura de GPU y el framework de CUDA C para programar el algoritmo. En el capítulo 5, se presenta la estrategia para implementar la FWI 2D elástica. El capítulo 6 contiene los resutados para todas las pruebas realizadas sobre el modelo CPS y el Overthrust 2D. El capítulo 7 presenta las conclusiones y las recomendaciones para el trabajo futuro.

1. Objetivos

Objetivo General

Desarrollar una estrategia de implementación de un algoritmo de inversión de onda completa (FWI) elástica 2D en el dominio del tiempo sobre una GPU.

Objetivos específicos

Modelar la propagación de ondas elásticas en GPU usando el método de diferencias finitas en un medio isótropo para generar datos sísmicos sintéticos para ser utilizados en la inversión 2D.

Establecer una estrategia de implementación de un algoritmo de inversión elástica 2D sobre una GPU para obtener los modelos de velocidad del subsuelo.

Evaluar la estrategia de implementación utilizando métricas que permitan medir el desempeño computacional en función de tiempo de ejecución, porcentaje de utilización de GPU y consumo de memoria.

2. Modelado elástico 2D

La propagación de ondas sobre determinado medio se ha convertido en una herramienta útil para obtener información relacionada con los parámetros físicos de interés en una determinada área de estudio. Las aplicaciones en distintos campos de estudio como las imágenes biomédicas Ginter et al. (2002); Kadlubiak et al. (2018), evaluación de materiales Moser et al. (1999) y el procesamiento de imágenes sísmicas Carcione et al. (2002) son sólo unos pocos ejemplos del potencial de investigación al cual un apropiado entendimiento de los fenómenos físicos asociados con la propagación de onda puede llevar.

El principal objetivo del procesamiento de imágenes sísmicas es alcanzar una imágen que muestre la composición del subsuelo en determinada zona de estudio lo mas cerca posible a la realidad. Actualmente, existen varios procedimientos para lograr este objetivo, entre ellos, la inversión de onda completa o (FWI) es ampliamente estudiada debido a su capacidad para generar modelos de alta resolución de la composición del subsuelo. La FWI se fundamenta en una comparación entre sismogramas obtenidos en una adquisición en campo y los sismogramas generados a través de la simulación de la propagación de onda o modelado de onda, por esta razón, lograr un modelado de onda adecuado es crucial para la implementación apropiada de FWI. Virieux et al. (2009)

Implementar el modelado numérico de onda involucra la selección de un operador de onda apropiado dependiendo del nivel de complejidad considerado en la zona de estudio, por tanto, es posible encontrar operadores como el acústico, elástico, viscoelástico y otros; la principal diferen-

cia entre ellos es la cantidad de parámetros que se utilizan para describir la misma zona de estudio, además de la complejidad en las relaciones entre dichos parámetros.

El operador acústico ha sido bastante estudiado desde hace mas de 30 años. Actualmente, el foco de la investigación a nivel mundial se da principalmente sobre versiones mas complejas como la elástica o viscoelástica. A nivel local en la Universidad Industrial de Santander (UIS), el grupo CPS ha logrado desarrollar conocimiento profundo en métodos de procesamiento de imágenes sísmicas con el operador acústico Abreo et al. (2017); Noriega et al. (2017), por este motivo, el grupo a propuesto nuevos objetivos de estudio en formulaciones mas complejas como es el caso del operador elástico considerado en este trabajo. Sanchez et al. (2017)

2.1. Formulación del modelado de onda elástico

El modelado de onda elástico considera que la composición de la tierra puede ser descrita utilizando tres parámetros llamados λ, μ and ρ; estos parámetros están relacionados por las ecuaciones de propagación elásticas presentadas por Virieux en la forma P-Sv Virieux (1986) en las ecuaciones 1 , 2 , 3 , 4 , 5.

$$\frac{\partial v_x}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z}\right), \tag{1}$$

$$\frac{\partial v_z}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z}\right), \tag{2}$$

$$\frac{\partial \sigma_{xx}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_x}{\partial x} + \lambda\frac{\partial v_z}{\partial z} + \varphi_{xx}, \tag{3}$$

$$\frac{\partial \sigma_{zz}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_z}{\partial z} + \lambda \frac{\partial v_x}{\partial x} + \varphi_{zz}, \tag{4}$$

$$\frac{\partial \sigma_{xz}}{\partial t} = \mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right), \tag{5}$$

donde x y z son coordenadas en el plano, v_x y v_z las componentes de velocidad, σ_{xx}, σ_{xz} y σ_{zz} son los esfuerzos y finalmente φ corresponde a la fuente que genera la onda propagada.

Existen varios tipos de ondícula para simular la fuente en la literatura, se hace especial mención en anáisis sísmico la ondícula Ricker, la Morlet, la gaussiana y otras ondas derivadas de la gaussiana Wang (2015), sin embargo, en este trabajo será considerada la ondícula Ricker ya que es la mas frecuentemente utilizada en el procesamiento por FWI y su ecuación se presenta en la expresión 6.

$$\varphi = (1 - 2\pi^2 f^2 t^2) e^{-\pi^2 f^2 t^2}. \tag{6}$$

Existen dos tipos de ondas en el modelado elástico, ondas compresionales y ondas de corte que viajan por el medio a distintas velocidades v_p and v_s, estas velocidades estan relacionadas con los parámetres elásticos; por este motivo es posible utilizar las relaciones mas comunes que existen entre λ, μ y las velocidades como se muestra en la ecuación 7 al resolver las ecuaciones elásticas.

$$V_p = \sqrt{\frac{\lambda + 2\mu}{\rho}}. \qquad V_s = \sqrt{\frac{\mu}{\rho}}. \tag{7}$$

Para entender mejor los resultados del modelado elástico, la figura 1 muestra la simulación en una zona de estudio de 535 [m] en dirección *offset* por 288 [m] en profundidad, la simulación usa una ondícula Ricker de frecuencia 20 [Hz], ubicada en el centro del modelo alrededor de 268 [m] en dirección *offset*, las capturas *a* y *b* muestran la propagación de onda como se espera. Sin embargo, en la captura *c* se observa un fenómeno numérico de reflexiones de la onda sobre los bordes del modelo que llevan a un mezcla de la onda con el tiempo como se muestra en la captura *d*; estos resultados son indeseados en un modelado adecuado y deben ser solucionados.

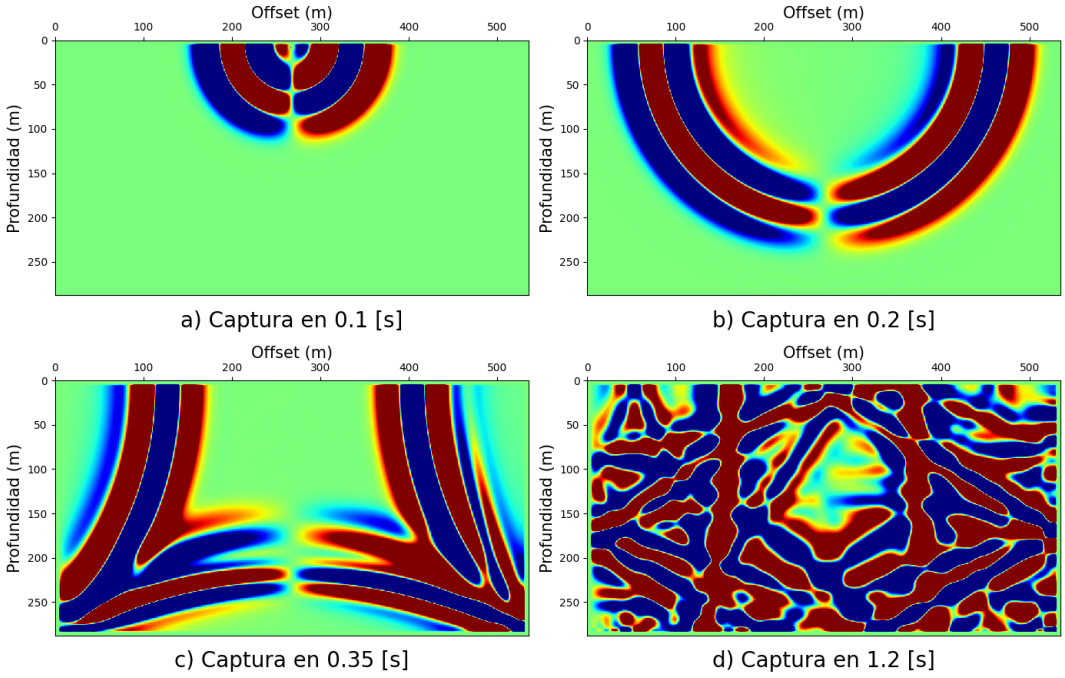

Figura 1. Capturas para la simulación del modelado de onda en una zona de estudio de 535 [m]x268 [m].

2.2. Comportamiento de bordes reflectivos e implementación de CPML

Cuando las ondas se propagan, el medio sobre el que se propaga se considera infinito Virieux et al. (2012), sin embargo, cuando se implementa modelado numérico el medio de propaga-

ción tiene unos bordes finitos que generan un comportamiento de reflexiones de onda indeseado, por tanto, para imitar el medio infinito computacionalmente se necesita definir unas zonas de disipación de energía (absorbentes) cerca de los bordes y prevenir la aparición de dichas reflexiones.

La figura 2 muestra un esquema resaltando las zonas de absorción de energía, la imágen muestra un modelo de dimensiones N_x-N_z, dentro de esta área se identifican 2 grandes espacios, el primero llamado *zona regular* y el segundo *zona absorbente*. Cuando se trabaja en la zona regular, las ondas se propagan siguiendo las expresiones presentadas 1 , 2 , 3 , 4 and 5; mientras que en la zona absorbente la propagación se rige por un sistema modificado de ecuaciones dependiendo del método de zonas absorbentes utilizado.

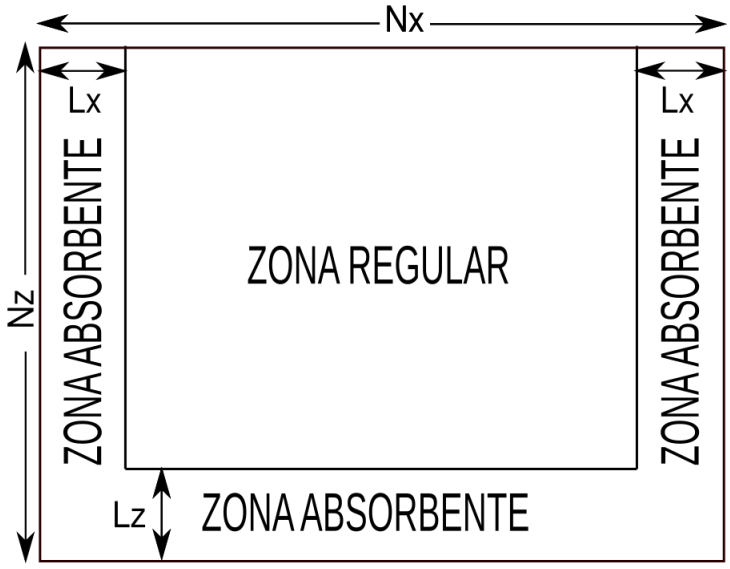

Figura 2. Modelo donde se localizan las zonas regular y absorbente.

Es importante resaltar que debido a que los zonas absorbentes son unos *añadidos* computacionales sobre las ecuaciones elásticas, siempre se quiere que en lo posible el impacto de estas zonas absorbentes sea el mínimo, esto se logra al disminuir las dimensiones L_x y L_z que correspon-

den con el grosor de la zona absorbente; sin embargo, la selección de este parámetro depende del desempeño del método usado para disipar energía.

Existen varios métodos para desarrollar esta tarea de absorción, entre estos se incluye límites de esponja, ABC híbrido, CPML y otros Gao et al. (2017), de entre ellos, Convolutional Perfectly Matched Layers (CPML) Komatitsch and Martin (2007) es ampliamente utilizado ya que ha demostrado ser muy efectivo en la reducción de energía comparado con otras opciones.

La implementación computacional del método CPML añade unas variables auxiliares a las expresiones de modelado originales como se muestra en las ecuaciones 8 , 9 , 10 , 11 y 12 .

$$\frac{\partial v_x}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z}\right) + \Omega_{xx} + \Omega_{xz}, \tag{8}$$

$$\frac{\partial v_z}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z}\right) + \Omega_{xz} + \Omega_{zz}, \tag{9}$$

$$\frac{\partial \sigma_{xx}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_x}{\partial x} + \lambda\frac{\partial v_z}{\partial z} + \varphi_{xx} + (\lambda + 2\mu)\Psi_{xx} + \lambda\Psi_{zz}, \tag{10}$$

$$\frac{\partial \sigma_{zz}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_z}{\partial z} + \lambda\frac{\partial v_x}{\partial x} + \varphi_{zz} + \lambda\Psi_{xx} + (\lambda + 2\mu)\Psi_{zz}, \tag{11}$$

$$\frac{\partial \sigma_{xz}}{\partial t} = \mu\left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x}\right) + \mu\Psi_{zx} + \mu\Psi_{xz}, \tag{12}$$

estas variables auxiliares se actualizan siguiendo las expresiones 13 , 14 , 15 , 16

$$\Omega_{xx}^{n} = b_x \Omega_{xx}^{n-1} + a_x \partial_x^{n+\frac{1}{2}} \qquad \Psi_{xx}^{n} = b_x \Psi_{xx}^{n-1} + a_x \partial_x^{n+\frac{1}{2}}, \tag{13}$$

$$\Omega_{zz}^{n} = b_z \Omega_{zz}^{n-1} + a_z \partial_z^{n+\frac{1}{2}} \qquad \Psi_{zz}^{n} = b_z \Psi_{zz}^{n-1} + a_z \partial_z^{n+\frac{1}{2}}, \tag{14}$$

$$\Omega_{xz}^{n} = b_z \Omega_{xz}^{n-1} + a_z \partial_z^{n+\frac{1}{2}} \qquad \Psi_{xz}^{n} = b_z \Psi_{xz}^{n-1} + a_z \partial_z^{n+\frac{1}{2}}, \tag{15}$$

$$\Omega_{zx}^{n} = b_x \Omega_{zx}^{n-1} + a_x \partial_x^{n+\frac{1}{2}} \qquad \Psi_{zx}^{n} = b_x \Psi_{zx}^{n-1} + a_x \partial_x^{n+\frac{1}{2}}, \tag{16}$$

donde x y z son coordenadas en el plano cartesiano, n es el tiempo discreto, a y b representan los coeficientes de atenuación dados por el método y su cálculo se realiza como se describe en las expresiones 17 , 18 y 19

$$a_u = \frac{du}{\kappa_u \left(d_u + \kappa_u \alpha_u\right)} \left(b_u - 1\right) \qquad u\{x,z\}, \tag{17}$$

$$b_u = e^{-\left(\frac{d_u}{\kappa_u} + \alpha_u\right)\Delta t} \qquad u\{x,z\}, \tag{18}$$

$$\alpha_u = \pi f_0 \frac{D}{L}. \tag{19}$$

Cuando se trabaja con modelado sísmico se realizan ciertas consideraciones, el parámetro $\kappa_u = 1$, L es el grosor de la zona CPML, D es el grosor de la zona CPML menos la distancia de penetración de la onda dentro de la zona CPML, f_0 es la frecuencia dominante dada por la ondícula usada como fuente.

Finalmente, la ecuación 20 muestra el cálculo del parámetro d_u donde para modelado elástico $R = 0.001$, L es el grosor de CPML, V corresponde a una velocidad del modelo normalmente elegida como la máxima, u es la distancia de penetración de la onda dentro de la zona CPML.

$$d_u = -\frac{3V}{2L} ln(R) \left(\frac{u}{L}\right)^2 \qquad u\{x,z\}. \tag{20}$$

La figura 3 muestra el resultado del modelado elástico con las mismas características de la simulación presentada en la figura 1, las capturas *a* y *b* muestran el mismo comportamiento que en la figura 1 , las últimas dos capturas son diferentes debido a que no aparecen las reflexiones en los bordes del modelo debido a la implementación del método CPML.

2.3. Discretización por diferencias finitas (FD)

Con las ecuaciones elásticas ya definidas como se muestra en 8, 9, 10, 11 y 12, el siguiente paso en el modelado elástico es la solución de este sistema de ecuaciones diferenciales; hay varios métodos numéricos para resolver las ecuaciones de propagación incluyendo las diferencias finitas, elementos finitos, volumenes finitos, métodos pseudoespectrales y otros Seriani and Oliveira (2020), entre ellos, el método de diferencias finitas (FD) es ampliamente utilizado debido a su facilidad de implementación y es el elegido para este trabajo.

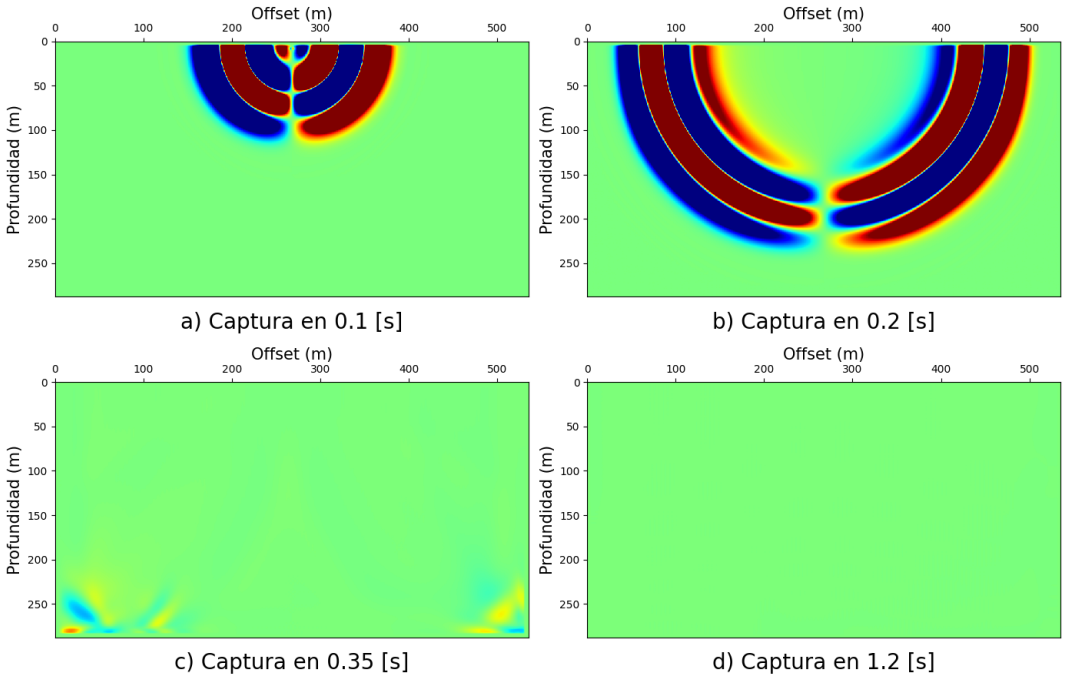

Figura 3. Capturas para la simulación del modelado de onda en una zona de estudio de 535 [m]x268 [m] con la aplicación del método CPML para evitar las reflexiones en los bordes.

Se hizo la elección de usar diferencias finitas de segundo orden en tiempo y cuarto en espacio ya que permite un balance aceptable entre eficiencia numérica y error de truncamiento pequeño. Las ecuaciones 21 , 22 y 23 son la expresiones de discretización para los operadores diferenciales que son aplicados a las ecuaciones de propagación. Para observar las fórmulas desarrolladas una vez aplicadas las diferencias finitas se pueden dirigir al anexo A.

$$\frac{\partial f^t_{x,z}}{\partial t} = \frac{1}{\Delta t}\left[f^{n+\frac{1}{2}}_{i,k} - f^{n-\frac{1}{2}}_{i,k}\right], \tag{21}$$

$$\frac{\partial f^t_{x,z}}{\partial x} = \frac{1}{\Delta x}\left[\frac{9}{8}\left(f^n_{(i+\frac{1}{2}),k} - f^n_{(i-\frac{1}{2}),k}\right) - \frac{1}{24}\left(f^n_{(i+\frac{3}{2}),k} - f^n_{(i-\frac{3}{2}),k}\right)\right], \tag{22}$$

$$\frac{\partial f_{x,z}^{t}}{\partial z} = \frac{1}{\Delta z}\left[\frac{9}{8}\left(f_{i,(k+\frac{1}{2})}^{n} - f_{i,(k-\frac{1}{2})}^{n}\right) - \frac{1}{24}\left(f_{i,(k+\frac{3}{2})}^{n} - f_{i,(k-\frac{3}{2})}^{n}\right)\right], \tag{23}$$

la expresión al lado izquierdo del igual es el operador diferencial en tiempo continuo respecto al tiempo en la expresión 21 y al espacio en las ecuaciones 22 y 23 ; las expresiones a la derecha son las equivalencias discretizadas en tiempo y en espacio para la implementación computacional donde n es el tiempo discreto, i y k son variables discretas de espacio; finalmente los pasos Δt, Δx y Δz son parámetros de diferencias finitas asociados con la precisión del método.

El proceso de discretización genera una distribución de cuadrícula 2D del medio donde todos las dimensiones físicas son adaptadas a espacios de memoria que tendrán información relacionada con los parámetros elásticos; en esta cuadrícula la longitud física y la profundidad se convierten en pequeños cuadritos de memoria N_x-N_z dependiendo de los pasos Δx y Δz seleccionados, como se muestra en la figura 4 ; para evitar efectos anisotrópicos en el modelado se suele definir que $\Delta x = \Delta z = \Delta h$.

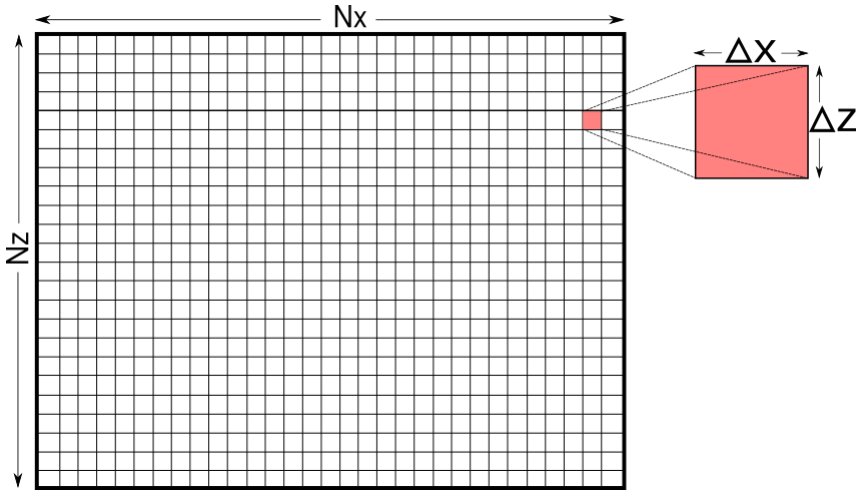

Figura 4. Distribución de cuadrícula del modelo discretizado.

Debido a la localización desplazada de los campos de velocidad y esfuerzos con sus derivadas en tiempo y espacio Moczo et al. (2007); Virieux et al. (2012), para resolver apropiadamente las ecuaciones de propagación por diferencias finitas centradas es necesario utilizar el esquema de malla intercalada. En esta distribución el valor de los campos de esfuerzos en una posición y tiempo determinado dependen de los campos de velocidad en posiciones distintas y viceversa como se muestra en la figura 5 .

La figura 5 muestra la distribución de los campos de velocidad y esfuerzo utilizando la malla intercalada; este esquema toma como referencia uno de esos pequeños cuadros resultantes de la cuadrícula de la discretización, los índices i y k son de espacio y n el índice de tiempo, la separación de los índices espaciales está dada por Δh y la separación temporal dada por Δt generando la estructura cúbica de la imágen.

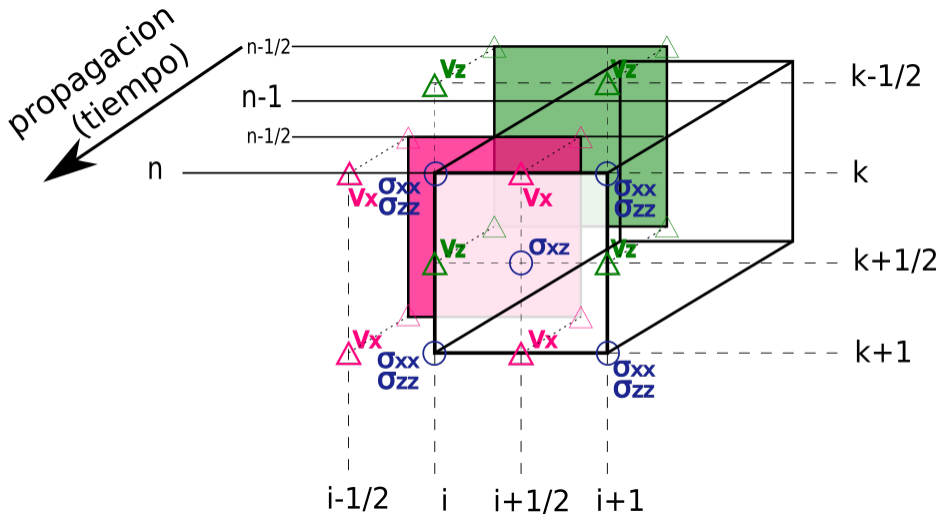

Figura 5. Malla intercalada en tiempo-espacio (3D) para la solución de los campos de velocidad y esfuerzo en las ecuaciones de propagación.

En la figura 5, se puede identificar fácilmente los campos de esfuerzo $\sigma_{xx}, \sigma_{zz}, \sigma_{xz}$ que se

calculan en el tiempo n, mientras los campos de velocidad se calculan por adelantado en el tiempo $n - \frac{1}{2}$; es también importante detallar que el campo de velocidad v_x esta desfasado del campo de velocidad v_z por $\frac{\Delta h}{2}$ tanto en longitud como en profundidad y estos a su vez están desfasados de los campos de esfuerzos, por esta razón es que se conoce como *intercalada*.

Debido a que el modelado elástico considerado es en 2 dimensiones, hay una forma muy común de representar la estructura de malla intercalada usada en la literatura y considera solo una *tajada* de tiempo para todos los campos y aquellos localizados en diferentes tajadas de tiempo son *proyectados* sobre la tajada elegida; estas proyecciones son normalmente obtenidas por técnicas de interpolación soportadas en la información de los campos de velocidad; la figura 6 muestra este esquema.

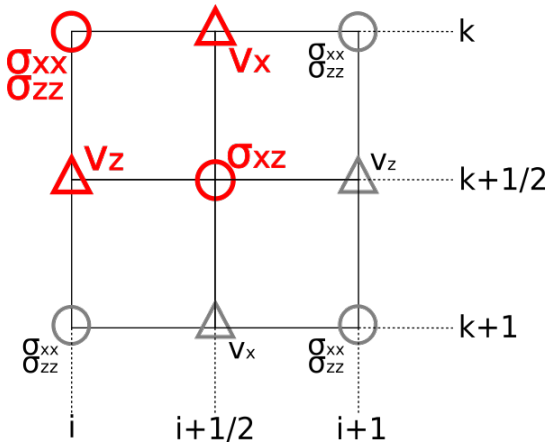

Figura 6. Representación usual 2D de la malla intercalada mostrando solo la distribución espacial considerando los campos de velocidad proyectados sobre los campos de esfuerzos.

Es importante resaltar que cuando se implementan las diferencias finitas existen algunas limitantes en la selección de los parámetros de discretización Δt y Δh; estos límites están relacio-

nados con la dispersión numérica máxima aceptada en la simulación dada por la expresión 24

$$\Delta h = \frac{V_{min}}{f_{max} n_\lambda}, \tag{24}$$

donde V_{min} corresponde a la velocidad mínima del modelo normalmente tomado como el menor valor de v_s, f_{max} es la frecuencia de la fuente usada en este caso la ondícula Ricker, n_λ corresponde al número de puntos por longitud de onda (ppw) elegido para ser utilizado en la simulación que es de 16 ppw en las simulaciones de este trabajo.

La otra limitante es el criterio de estabilidad numérica que depende de la selección del esquema de diferencias finitas, con base en la información presentada por Lines en la nota corta Lines et al. (1999) que resume las fórmulas de criterio de estabilidad, para diferencias finitas de segundo orden en tiempo y cuarto en espacio la expresión 25 controla este criterio.

$$\Delta t = \sqrt{\frac{3}{8}} \frac{\Delta h}{V_{max}}, \tag{25}$$

donde Δh es el paso espacial and V_{max} es la máxima velocidad del modelo que en este trabajo será tomada como el valor máximo de la velocidad v_p.

3. Inversión de onda completa (FWI)

Cuando las ondas viajan a través de un medio, ellas interactán con diferentes tipos de elementos que componen dicho medio, esas interacciones asociadas a las distintas propiedades de los elementos generan desviaciones en la dirección de la onda al propagarse por efecto de refracciones, reflexiones y otros fenómenos físicos que ocurren por esas interacciones. Virieux et al. (2017) Los resultados de la propagación de onda y las interacciones con el medio pueden ser recuperados en los conocidos sismogramas como se muestran en la figura 7.

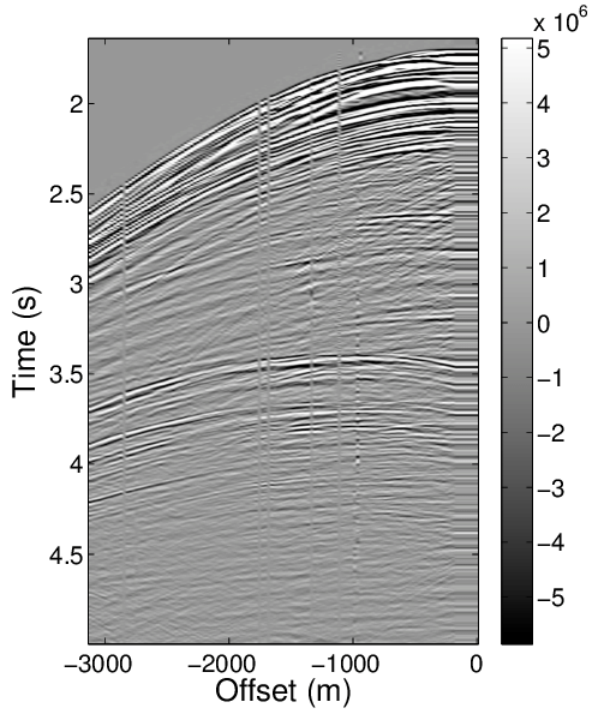

Figura 7. Ejemplo de un sismograma real, tomado de Duarte et al. (2014).

El sismograma recolecta los resultados de esas interacciones de la onda en la profundidad de la tierra por medio de mediciones realizadas en la superficie utilizando los geófonos; la idea en

FWI es que un procesamiento adecuado, análisis e interpretación de los sismogramas puedan dar razón de la composición del subsuelo y debido a esto los sismogramas son parte fundamental en el procesamiento de imágenes sísmicas incluyendo la inversión de onda completa.

La formulación elástica para el modelado introdujo los parámetros que describen la tierra λ, μ, ρ o los más comunes ρ, V_p, V_s, matematicamente estos parámetros pueden representarse como un vector $\mathbf{m}$; los sismogramas son el resultado de recolectar información sobre las interacciones de la onda al viajar y recopilan información sobre esos parámetros que modelan la tierra, por tanto, los sismogramas pueden ser representados como una función de los parámetros $d(m)$.

La inversión de onda completa (o FWI) Tarantola (1984, 1986); Gauthier et al. (1986) es básicamente un procedimiento para generar imágenes de alta resolución de la composición del subsuelo con base en la suposición de un modelo inicial de parámetros elásticos que es actualizado de forma iterativa hasta que alguna condición de control es alcanzada. Para este caso, la condición depende de la comparación realizada entre un sismograma obtenido en una adquisición de campo que a menudo se conoce como *dato verdadero* $d_{true}(m)$ y el sismograma que viene de una simulación de modelado de onda o *dato modelado* $d_{mod}(\hat{m})$.

La condición de control o mas formalmente *función de costo* pueden tener varias formas de definición, sin embargo, se usará la definición de minimización por mínimos cuadrados Tarantola (2005) ya que es ampliamente reconocida en el procesamiento sísmico. La ecuación 26 muestra la definición de la función de costo por mínimos cuadrados donde $d_{true}(m)$ viene de un sismograma obtenido sobre un modelo con los parámetros reales m, la expresión $d_{mod}(\hat{m})$ viene de un

sismograma modelado obtenido de un modelo con parámetros actualizados $\hat{m}$.

$$\Phi(\hat{m}) = \arg\min_{\hat{m}} ||d_{mod}(\hat{m}) - d_{true}(m)||_2^2 \tag{26}$$

El núcleo de FWI considera un modelo con determinado valor inicial en los parámetros elásticos que será actualizado de forma contínua utilizando el modelado elástico generando a la vez el sismograma modelado; en cada iteración, al comparar el sismograma modelado con el verdadero se obtiene un conjunto de residuales que se convierten a su vez en el insumo para el procedimiento de actualización del modelo inicial y de esta manera proseguir con el proceso iterativo. La figura 8 muestra a la izquierda un ejemplo de sismograma verdadero sintético obtenido por modelado sobre un modelo con los parámetros reales, la imágen de la derecha es un ejemplo de sismograma modelado obtenido por modelado elástico sobre un modelo inicial.

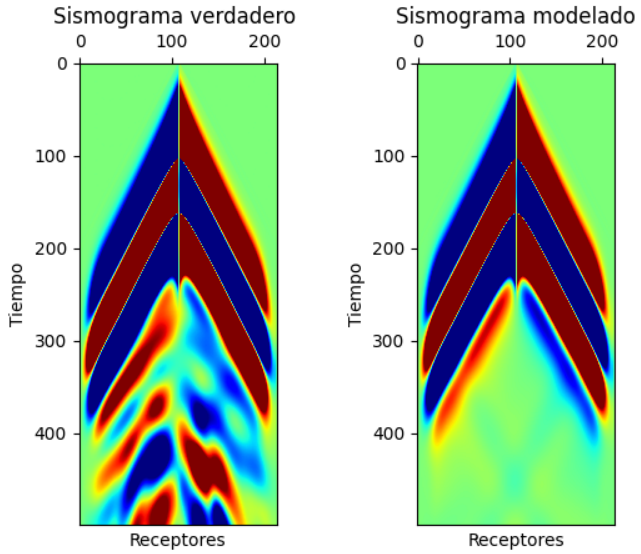

Figura 8. Ejemplos de sismogramas sintéticos, a la izquierda, el sismograma verdadero, a la derecha, el sismograma modelado.

Al final, se espera que gracias a la constante actualización de los parámetros elásticos modelados $\hat{m}$ que el sismograma modelado $d_{mod}(\hat{m})$ tienda a mostrar el mismo comportamiento de trazas que el sismograma verdadero $d_{true}(m)$ y por tanto permita inferir que los parámetros elásticos modelados están muy cerca de los reales.

Se ha presentado el procedimiento de como la FWI apoyada en el modelado de onda en un proceso de actualización iterativo puede ayudar a la exracción de la información de los parámetros elásticos contenida en los sismogramas; sin embargo, no es claro todavía como el modelo inicial es actualizado en cada iteración FWI. Ya que la FWI es básicamente un problema de optimización no lineal existen varios métodos numéricos para desarrollar esta tarea incluyendo el método de *steepest descent* o gradiente descendente Tarantola (2005); Nocedal and Wright (2006), gradiente conjugado Segovia Spadini et al. (2016), *Gauss-Newton* Chen and Sacchi (2018) y otros Virieux and Operto (2009) con diferentes ventajas y desventajas. En este trabajo será utilizado el gradiente descendente cuya formulación se presenta en la ecuación 27

$$\hat{m}^{k+1} = \hat{m}^{k} - \alpha \nabla f_m, \tag{27}$$

donde $\hat{m}^{k+1}$ es el modelo de parámetros eláticos actualizado en la iteración de FWI $k+1$, $\hat{m}^{k}$ es el modelo de parámetros elásticos en la última iteración k, el parámetro del método α es el paso que se asocia con la magnitud hacia la dirección de actualización y ∇f_m es el gradiente de parámetros elásticos que indica la dirección de actualización para minimizar la función de costo.

3.1. Ecuaciones adjuntas y retro propagación

En este punto se han presentado todos los componentes para implementar FWI, sólo una definición falta y es como calcular el gradiente en la ecuación 27 , para ello es ampliamente utilizado en procesamiento sísmico el método de estado adjunto propuesto por Plessix Plessix. (2006). A pesar de que esta metodología es completa por si sola para la implementación de la teoría de estado adjunto y obtener los gradientes necesarios en FWI, existe un reporte de investigación en el grupo CPS que contiene una interpretación de todo el fundamento matemático y lo presenta en una forma más adecuada para ingenieros Wesdorp (2018)

Al considerar la teoría de estado adjunto es posible calcular los gradientes al relacionar el modelado elástico con el nuevo operador de estado adjunto. En esta teoría cada campo de velocidad y esfuerzo de la formulación elástica tiene un campo adjunto asociado, por ejemplo el campo v_x tiene campo adjunto correspondiente que en este trabajo se denominará l_1, por lo tanto se tiene que $v_x \rightarrow l_1$, $v_z \rightarrow l_2$, $\sigma_{xx} \rightarrow l_3$, $\sigma_{zz} \rightarrow l_4$ and $\sigma_{xz} \rightarrow l_5$.

Al establecer la existencia de los campos adjuntos, hay una definición de los operadores de propagación adjuntos o ecuaciones de retro propagación presentadas en las ecuaciones 28 , 29 , 30 , 31 y 32 , estas expresiones se obtienen al desarrollar la metodología expuesta en el reporte de investigación

$$\rho \frac{\partial l_1}{\partial t} = \frac{\partial \left((\lambda + 2\mu) l_3 \right)}{\partial x} + \frac{\partial \left(\lambda l_4 \right)}{\partial x} + \frac{\partial \left(\mu l_5 \right)}{\partial z}, \tag{28}$$

$$\rho \frac{\partial l_2}{\partial t} = \frac{\partial \left(\lambda l_3 \right)}{\partial z} + \frac{\partial \left((\lambda + 2\mu) l_4 \right)}{\partial z} + \frac{\partial \left(\mu l_5 \right)}{\partial x}, \tag{29}$$

$$\frac{\partial l_3}{\partial t} = \frac{\partial l_1}{\partial x}, \tag{30}$$

$$\frac{\partial l_4}{\partial t} = \frac{\partial l_2}{\partial z}, \tag{31}$$

$$\frac{\partial l_5}{\partial t} = \frac{\partial l_1}{\partial z} + \frac{\partial l_2}{\partial x}. \tag{32}$$

Con esta información de la retro propagación junto con la del modelado es posible ahora construir las expresiones para el cálculo de los gradientes.

Es importante mencionar que ya que la retro propagación se puede considerar otra forma de realizar modelado de onda pero utilizando un nuevo conjunto de campos, su codificación computacional involucra la solución de ecuaciones por diferencias finitas de la misma forma como fue realizado en el caso del modelado, por tanto la formulación respecto a CPML y discretización debe ser adaptada a las nuevas fórmulas de campos adjuntos. A pesar de ello si hay una gran diferencia, la dirección del tiempo para resolver las ecuaciones en la retro propagación es contraria a la establecida en el modelado.

En el modelado, los cálculos se realizan iniciando con el tiempo cero hasta un determinado tiempo final mientras en la retro propagación se inicia desde el tiempo final y se retrocede hasta llegar al tiempo cero; la figura 9 presenta lo explicado de forma gráfica y que es importante entender para poder implementar una adecuada retro propagación.

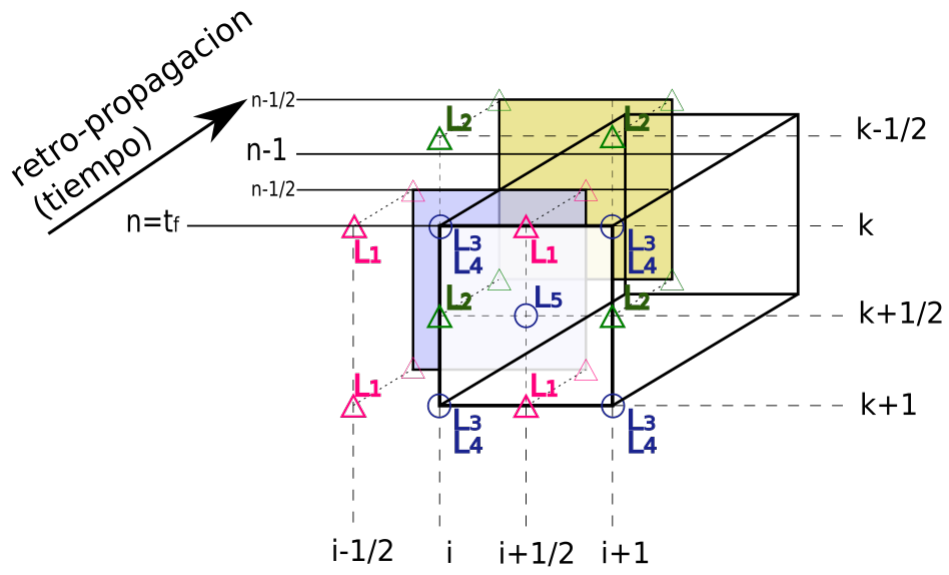

Figura 9. Malla intercalada tiempo-espacio (3D) para la solución de los campos adjuntos.

Las fórmulas discretizadas para las ecuaciones adjuntas usando diferencias finitas de segundo orden en tiempo y cuarto en espacio se presentan en el apéndice B.

3.2. Cálculo de los gradientes de parámetros elásticos

Una vez definidos los operadores de modelado y la retro propagación y su implementación computacional, es posible proceder con el cálculo de los gradientes. De acuerdo al procedimiento de estado adjunto en el reporte de investigación y considerando que la expresión ∇f_m debe relacionar el gradiente de cada parámetro elástico $\frac{\partial f_m}{\partial \rho}, \frac{\partial f_m}{\partial \lambda}, \frac{\partial f_m}{\partial \mu}$; el resultado lleva a tres diferentes fórmulas para la obtención de los gradientes como se muestra en las ecuaciones 33 , 34 y 35.

$$\frac{\partial f_m}{\partial \rho} = -\int_0^T \left(\overrightarrow{l_1} \frac{\partial \overrightarrow{v_x}}{\partial t} + \overrightarrow{l_2} \frac{\partial \overrightarrow{v_z}}{\partial t} \right) dt, \tag{33}$$

$$\frac{\partial f_m}{\partial \lambda} = -\int_0^T \left(\overrightarrow{l_3} + \overrightarrow{l_4}\right)\left(\frac{\partial \overrightarrow{v_x}}{\partial x} + \frac{\partial \overrightarrow{v_z}}{\partial z}\right) dt, \tag{34}$$

$$\frac{\partial f_m}{\partial \mu} = -\int_0^T \left(2\overrightarrow{l_3}\frac{\partial \overrightarrow{v_x}}{\partial x} + 2\overrightarrow{l_4}\frac{\partial \overrightarrow{v_z}}{\partial z} + \overrightarrow{l_5}\left(\frac{\partial \overrightarrow{v_x}}{\partial z} + \frac{\partial \overrightarrow{v_z}}{\partial x}\right)\right) dt. \tag{35}$$

Al observar estas expresiones es claro el porqué se debe contar con campos procedentes del modelado v_x, v_z. Ya que las ecuaciones de los gradientes contiene derivadas parciales, deben ser resueltas utilizando diferencias finitas y por tanto aplica en este caso también aplica la teoría sobre discretización. En este punto también es clara la importancia de implementar un adecuado modelado de onda para lograr resultados de la FWI apropiados.

En la mayoría de las adquisiciones en campo, se realizan múltiples pruebas en distintas localizaciones intentando recolectar la mayor cantidad de información de la zona de estudio. Por tanto, en la mayoría de las implementaciones FWI se suele contar con varios sismogramas verdaderos tomados con la fuente ubicada en distintas posiciones que puede contribuir con la mejora en la reconstrucción del modelo final.

Cuando se implementa la metodología de FWI elástica cada sismograma verdadero se asocia con un modelado de onda que debe ser acompañado con una retro propagación y un cálculo de gradiente de la forma descrita, al final se obtiene un gradiente global para cada parámetro elástico y se usa en las ecuaciones de actualización. Por este motivo implementar FWI utilizando toda la información disponible de una adquisición puede ayudar a lograr mejores resultados en la definición de las imágenes de los parámetros, sin embargo, existe un aumento en tiempo de ejecución debido

a la mayor cantidad de información procesada que debe tenerse en cuenta si se quiere alcanzar resultados adecuados en tiempos realistas.

3.3. FWI es un problema *mal puesto*

Con toda la teoría presentada, se puede pensar que la implementación computacional de la FWI es algo sencillo de lograr una vez definidas las entradas, sin embargo, la FWI presenta varios problemas por tratar de los cuales el mas conocido es su condición de problema *mal puesto*.

La FWI es un método de minimización local no-lineal que intenta encontrar el minimo global de una determinada función, en este caso mínimos cuadrados, a través de un ajuste de residuales entre un sismograma verdadero y uno modelado; el problema mal puesto viene del hecho que múltiples modelos finales pueden ser considerados una respuesta que ajusta los datos de la comparación.

La figura 10 esquematiza este problema gráficamente, aquí una hipotética función de costo es trazada donde existen dos mínimos locales y uno global, si el método parte de un modelo inicial como V_{ini1}, la inversión intentará ir hacia el mínimo mas cercano que en este caso es uno local, los datos pueden llegar a tener ajuste en este mínimo sin embargo no corresponde al mínimo global buscado; de otra parte si se inicia con el modelo inicial V_{ini2} se puede suponer que tenderá sobre el mínimo global esperado.

La última afirmación debe ser tomada en cuenta pues deja claro la importancia de la apropiada selección del modelo inicial cuando se ejecuta la FWI, sin embargo, no existe una receta mágica para facilitar este proceso. A pesar de esto, existen algunas recomendaciones para la construcción de modelos iniciales entre ellas el uso de tomografía de reflexión, tomografía de primeros

arribos, estereotomografía y otros Virieux and Operto (2009) que pueden ser útiles.

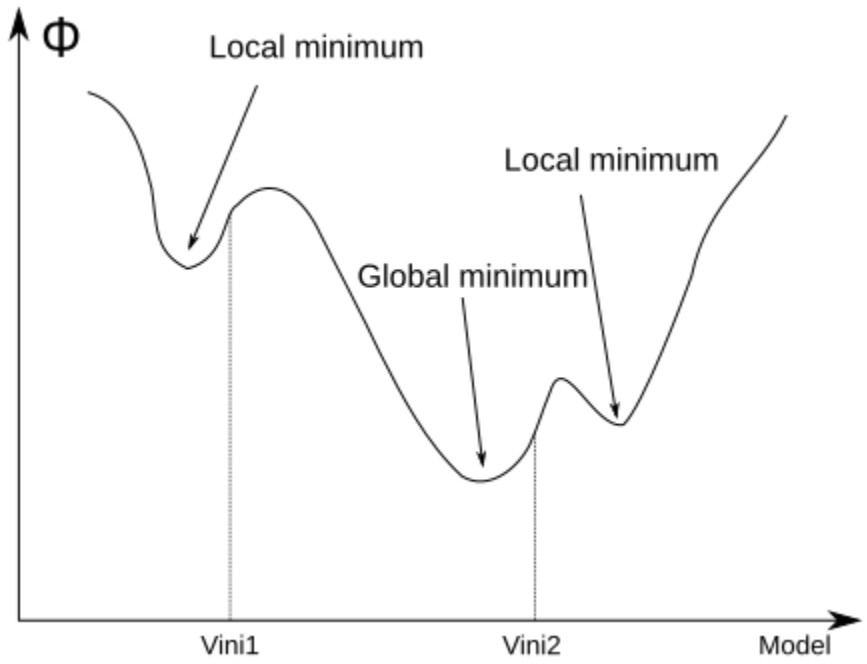

Figura 10. Función de costo hipotética y localización de mínimos locales y globales, el problema mal puesto sucede porque un mal modelo inicial puede llevar hacia un mínimo local.

4. Computación de alto desempeño (HPC)

Debido a la gran cantidad de datos por procesar y la complejidad de las relaciones entre los datos, la FWI elástica se considera un procedimiento de alta demanda de recursos computacionales; es tan costoso que para ponerlo en contexto, toda la teoría para implementar FWI fue desarrollada en los años 80 y algunos intentos de implementación fueron desarrrollados Crase et al. (1990); Crase et al. (1992) pero fue hasta hace unos 20 años que la evolución en hardware y el surgimiento de las unidades de procesamiento gráfico (GPU por sus siglas en inglés) ha permitido lograr implementaciones computacionales realistas.

Ya que la implementación de FWI involucra la ejecución de modelado, retro propagación y cálculo de gradientes y esta operación se repite tantas veces como fuentes disponibles, el método es costoso tanto en términos de consumo de memoria como de tiempo de ejecución necesario para alcanzar resultados apropiados. Debido a lo mencionado surge la necesidad de explorar nuevas estrategias de codificación computacional y evolucionar de los viejos paradigmas en programación para tratar con el problema de alto consumo de recursos de FWI.

El método de FWI involucra la aplicación de múltiples tareas repetitivas que en la programación secuencial, que fue la más ampliamente utilizada en las etapas tempranas de la ciencia computacional, no puede ser ejecutado eficientemente y por tanto la dificultad de obtener implementaciones precisas y realistas de FWI. Sin embargo, la evolución en la arquitectura de la GPU junto con la rapida expansión del nuevo paralelismo inherente a estos dispositivos ha permitido expandir las capacidades de la programación secuencial y el surgimiento de un nuevo paradigma

de programación conocido como *computación heterogégenea*; en pocas palabras, la computación heterogénea es la unión de los potenciales de la programación secuencial (control de ejecución) y paralela (ejecución de tareas con alta carga computacional) para resolver aplicaciones demandantes.

Estas herramientas de actualidad, la computación heterogénea junto con la evolución de la GPU y la introducción de los clusteres de computo; están siendo utilizadas por los investigadores en diversas áreas como la meteorología, la energía, simulaciones climáticas, modelado molecular entre otras y esto a llevado al desarrollo de la nueva área de investigación conocida como *computación de alto desempeño* o (HPC).

4.1. Computación paralela

La computación paralela es un paradigma de programación que se fundamenta en la idea de que un problema grande puede ser dividido en varios problemas pequeños que pueden ser resueltos simultáneamente. Si tomamos en cuenta las operaciones repetitivas que encontramos al implementar FWI, la aplicación de este paradigma permite no solamente resolver las tareas mas rápidamente sino lograr una mejor distribución de los recursos computacionales.

La figura 11 presenta un esquema de la programación secuencial, aquí el código es dividido en varias instrucciones que son ejecutadas una por una siguiendo determinado orden de ejecución; para ejecutar la sentencia actual, la anterior ya debe haber finalizado y la próxima debe esperar hasta que la actual finalice.

De esta forma, existe un impacto en el tiempo de ejecución total debido a la suma de los retardos de cada sentencia al ser ejecutada y en el caso donde aparecen sentencias repetitivas la

penalidad puede verse aumentada, de forma similar, los recursos computacionales no son utilizados eficientemente debido a que un bloqueo del espacio de memoria cuando una instrucción se ejecuta genera una demora en el proceso completo; el principal beneficio de la programación secuencial radica en el control del programa en cada instrucción ejecutada que puede ayudar en la detección de errores.

Figura 11. Programación secuencial donde las instrucciones son ejecutadas una a una en línea, la última instrucción debe esperar a que las anteriores terminen. Tomado de Cheng et al. (2014).

En la programación paralela el código completo es dividido en varias instrucciones más pequeñas también, sin embargo, es posible ejecutar de forma simultánea sentencias repetitivas o aisladas separadas del hilo (thread) principal que controla la ejecución del programa completo, sin embargo la interacción entre ellas esta a cargo del hilo principal, además cada una de esas instrucciones separadas tienen un propio espacio de memoria para operar evitando los mencionados retrasos por bloqueo de memoria. La figura 12 muestra el esquema de este paradigma de programación.

A pesar de las ventajas del paralelismo también tiene desventajas, entre ellas se puede mencionar la incapacidad para implementar estructuras de control avanzado en los hilos ejecutados en paralelo, estos hilos son pensados para tratar con instrucciones simples pero de alta carga compu-

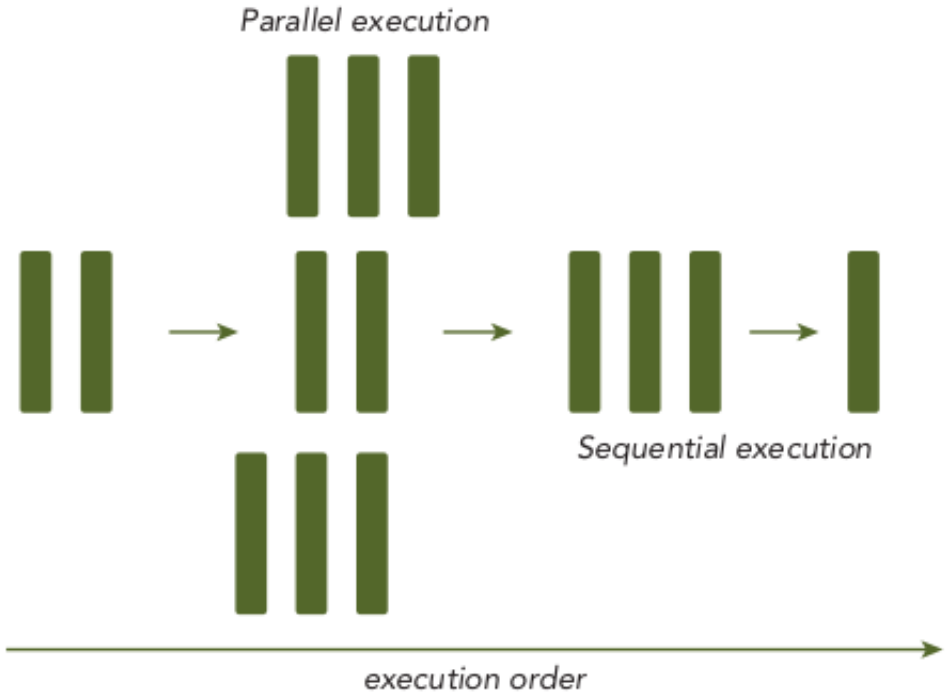

Figura 12. Programación paralela donde múltiples instrucciones sencillas son ejecutadas de forma simultánea junto con las secuenciales. Tomado de Cheng et al. (2014).

tacional que permitan tratar con las instrucciones mas pesadas del programa, de esta forma se deja el control al hilo principal en la ejecución secuencial, ya que estas operaciones pueden reducir su desempeño; otra desventaja radica en el hecho de que los hilos en paralelo no pueden existir por si solos, ellos son siempre deben ser controlados por el hilo principal y es este el que maneja la asignación de memoria, sincronización de datos y otras tareas de comunicación entre hilos.

Al final, ambos paradigmas muestran ventajas y desventajas, el reto es encontrar un balance en como usar los beneficios combinados de las estrategias con el objeto de resolver problemas computacionales complejos como lo es la implementación de FWI.

4.2. Computación heterogénea

La computación heterogénea busca aprovechar lo mejor de la programación secuencial y paralela con el objetivo de crear implementaciones computacionales poderosas para resolver problemas complejos. La principal preocupación en este paradigma es como interconectar o crear comunicaciones entre las partes de código que se ejecutan de forma secuencial con las que se ejecutan en paralelo.

Es usual encontrar que en sistemas con arquitectura x64 las instrucciones se ejecutan en la unidad central de procesamiento (CPU), a ella se le asigna el control, administración de memoria y tareas pesadas; sin embargo, con el incremento en las demandas computacionales las CPU han tenido que evolucionar de un solo núcleo a multinúcleo/multihilos para tratar con la carga computacional actual y hoy es posible encontrar CPU con alrededor de 72 hilos.

Esos núcleos/hilos en la CPU pueden operar sobre múltiples instrucciones de forma simultánea y pueden ser usados en programación paralela para explotar estos beneficios a través de la implementación de instrucciones MPI por ejemplo, sin embargo, esos hilos tienen una capacidad de control computacional superior que en algunos casos exceden la necesidad de determinada instrucciones simples que a menudo se encuentran en programas complejos, añadiendo al hecho que en algunos casos 72 hilos pueden no ser suficientes en todas las aplicaciones lleva a pensar que utilizar paralelismo en CPU puede ser inadecuado en algunos contextos.

Las GPU por su parte fueron concebidas para tratar con la alta carga que se encuentra en el procesamiento de imágenes una tarea que se caracteriza por su alto nivel de paralelismo y por tanto

su principal foco ha sido en la industria de los videojuegos de forma casi exclusiva, a pesar de ello, la inherente capacidad de la GPU debido a la gran cantidad de hilos no-especializados disponibles es ahora explotada como una opción real para enfrentar la carga computacional que se encuentra en diferentes áreas de investigación como las predicciones climáticas, simulaciones moleculares, FWI y muchas más.

La computación heterogénea Arora (2012) busca explotar la sinergia constructiva que puede ser creada al unir el poder de la CPU en administración y control con la potencia de ejecutar instrucciones que generan alta carga computacional de la GPU llevando a un incremento en el desempeño general comparado con el que se puede alcanzar con cada tecnología por separado. La figura 13 condensa la forma como ocurre la distribución de las cargas en CPU y GPU cuando se ejecuta código bajo este paradigma.

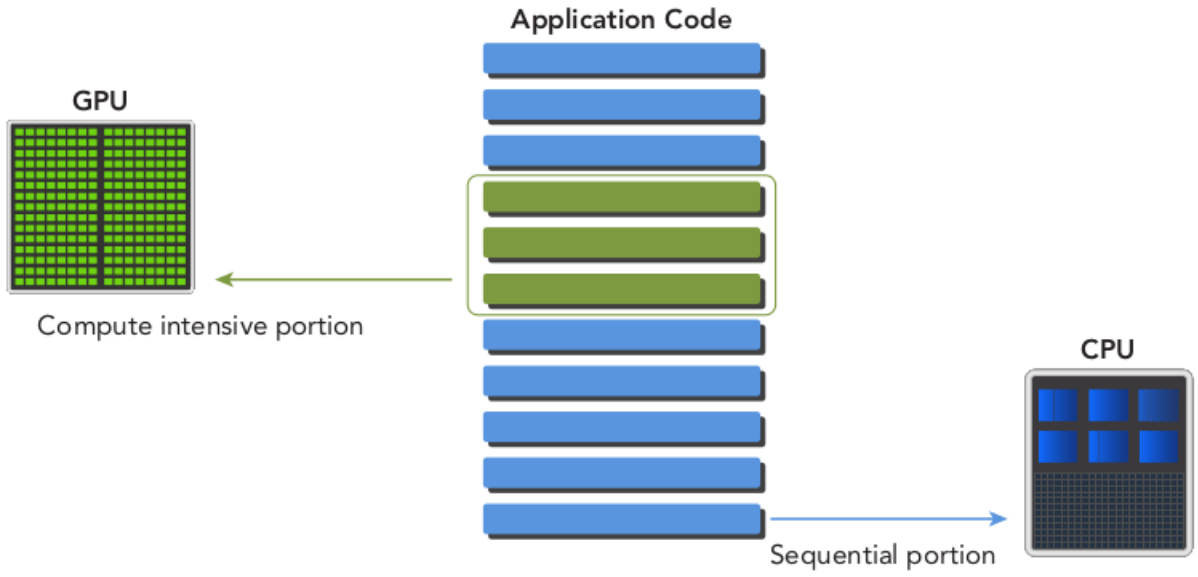

Figura 13. Implementación del paradigma heterogégeno. Tomado de Cheng et al. (2014).

De forma similar al framework MPI utilizado en la implementación de paralelismo en CPU directamente en sus núcleos, existen otros frameworks para lograr una interacción apropiada entre GPU-CPU e implementar paralelismo, entre ellos CUDA-C y OpenCL son los mas ampliamente utilizados, el primero es una opción propietaria de Nvidia mientras el último es una opción libre implementada principalmente en las GPU de AMD; debido a que se cuenta con hardware de Nvidia y su framework está optimizado, la opción utilizada será CUDA -C.

4.3. Unidades de procesamiento gráfico (GPU)

La potencia de implementar paralelismo utilizando GPU radica en el número de hilos disponibles para ejecutar muchas instrucciones de forma simultánea y administrar esos hilos para resolver determinadas tareas.

Teniendo en cuenta que las GPU Nvidia cuentan con una arquitectura única que se fundamenta en los llamados *streaming multiprocessors* o SM Cheng et al. (2014) estos engloban los núcleos CUDA que contienen los hilos para ejecutar las instrucciones; dentro de los SM existen diferentes niveles de memoria incluyendo *shared*, local, varios tipos de caché y registros; estos espacio de memoria permiten a los diferentes hilos la comunicación exitosa y la transferencia de datos cuando las instrucciones están corriendo.

Cada núcleo CUDA desarrolla distintos tipos de operaciones y por tanto es posible encontrar unidades FP32 y INT32 que ejecutan instrucciones de punto flotante y enteras, el número y distribución de estas unidades en la GPU depende del estado actual de la arquitectura; de este modo Nvidia ha estado en constante desarrollo y actualización de sus equipos añadiendo nuevas funcionalidades.

Las arquitecturas Nvidia han pasado desde Fermi Nickolls and Dally (2010) y Kepler a las mas actualizadas Turing y Ampere, la evolución ha incluido muchas características nuevas para mejorar la potencia de cálculo y soportar las tecnologías emergentes como inteligencia artificial y deep learning; por esta razón en el hardware actual, se puede encontrar fácilmente nuevos núcleos conocidos como los *tensor cores* especializados en operaciones matriciales y los *RT cores* enfocados en el trazado de rayos. Todo este hardware que compone un SM es mostrado en el esquema de la figura 14.

En este punto, es mas clara la razón por la cual las GPU son poderosas herramientas cuando se requiere la implementación de paralelismo, sin embargo, el poseer las herramientas no es suficiente para resolver los problemas; es necesario poner a trabajar las herramientas estableciendo estrategias para comunicación y sincronización de los diferentes hilos trabajando en tareas separadas y recopilando sus resultados para lograr resolver el problema principal. Para lograr esto se debe utilizar en ambiente de programación como lo es el framework CUDA-C.

4.4. Nvidia CUDA C framework

CUDA significa arquitectura de computo de dispositivo unificada que se refiere a framework para la programación en paralelo utilizando GPU bajo el paradigma de computación heterogénea; fue desarrollado por Nvidia como un una herramienta propietaria; comúnmente se escucha nombrarlo como CUDA C ya que intenta mantener la mayoría de la estructura de programación en C cuando se codifica pero aplica el paralelismo en GPU. Debido a que el paralelismo puede ser implementado utilizando otros lenguajes de programación, CUDA C solamente es una opción *wrapper*, existen así mismo otras opciones incluyendo PyCUDA, CUDA con Java y Fortran, sin

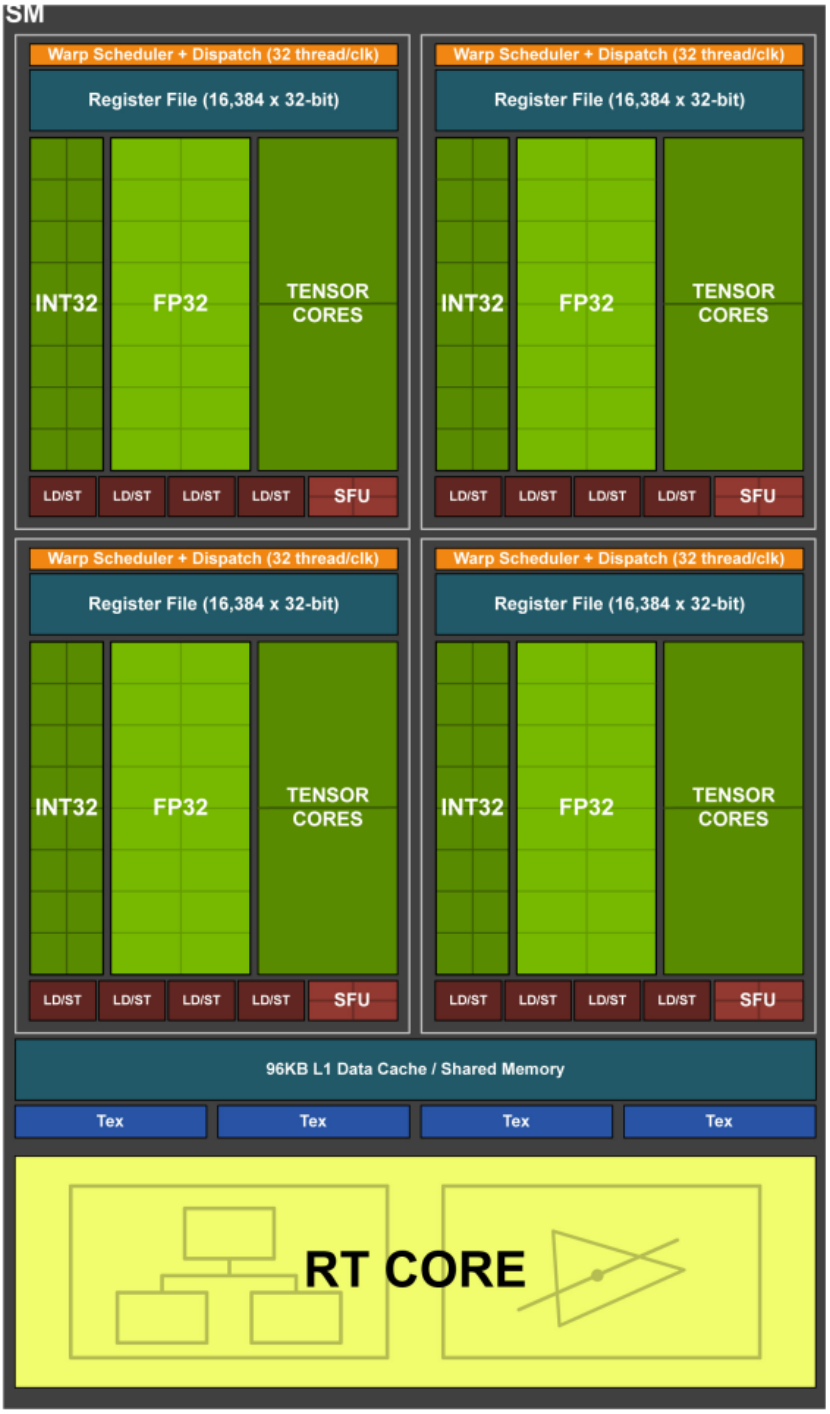

Figura 14. Arquitectura GPU Nvidia Turing. Tomado de NVIDIA (2018).

embargo, CUDA C es la opción mas extendida.

CUDA utiliza arquitectura SIMT (Single Instruction Multiple Thread) donde básicamente la instrucción es ejecutada simultáneamente por todos los hilos que componen el *warp* que se refiere a un grupo de 32 hilos, cada uno de esos hilos tienen una determinada cantidad de recursos incluyendo registros, contador de direcciones y espacio de memoria para procesar sus datos. Esta definición de *warp* es importante debido a que la ejecución de los hilos en GPU específicamente en el SM es dividida por el tamaño del warp dependiendo de los recursos de hardware disponible.

En el framwork CUDA se han definido dos áreas principales de código, la que será ejecutada en CPU (Host) y la que se planea paralelizar en GPU (Device), aquí la porción de cd́digo a paralelizar se escribe como funciones o *kernels*; la mayoría del código que se ejecuta en CPU no varia mucho dÂ´e la sintaxis utilizada en C, sin embargo, el uso de una sintaxis propietaria debe ser considerada cuando se están escribiendo y ejecutando Kernels.

El proceso de ejecución de un kernel en GPU incluye no solo la definición del código paralelo dentro del kernel sino la asignación de recursos de hardware de la GPU para realizar la ejecución apropiadamente, entre esos recursos se puede mencionar la definición de variables, el uso de memoria compartida pero también la definición de cuantos hilos serán necesarios.

Para desarrollar esta tarea CUDA usa la estrategia conocida como el diseño de hilos donde el paralelismo es implementado en el kernel con la definición de dos estructuras, el *thread/block* y el *grid*. El primero se relaciona con el número de hilos que un bloque contiene y en la última se especifica cuanto bloques son necesarios para ejecutar un kernel de forma simultánea; la combinación de estas dos definiciones crean el *layout* o diseño para implementar paralelismo en la

GPU.

La figura 15 muestra el diseño lógico de una implementación, la interpretación en este caso es que el código en el kernel es distribuido en el grid que contiene los bloques y a su vez estos bloques contienen los hilos de ejecución. Es interesante resaltar que el kernel se ubica inicialmente en la CPU que indica que el lanzamiento se realiza desde el host pero la ejecución sucede en la GPU; este comportamiento viene del paradigma de la computación heterogénea donde la parte secuencial se localiza en CPU y le es dado el control del algoritmo, las transferencia de información e implementación de estructuras complejas mientras la porción de alto costo computacional se le cede a la mas capaz GPU.

La definición apropiada del diseño de hilos es un tarea importante ya que se controla no solo cuantos hilos son utilizados sino también determina la cantidad de memoria por hilo y bloque, además una selección apropiada puede llevar a un uso mas eficiente del potencial de los SM en la GPU. CUDA permite la definición de diseños 1D, 2D y 3D sin embargo la elección de cual usar depende de la aplicación que será ejecutada.

En la figura 16, se muestra un ejemplo de diseño de hilos, se considera un kernel que correrá sobre 16 hilos en GPU, la opción *a* es un diseño 1D-1D porque solo hay un bloque y los hilos están dispuestos linealmente, la opción *b* tiene una distribución 1D-2D similar a la anterior pero esta vez los hilos se disponen en dos filas dentro del mismo bloque, la opción *c* es una distribución 1D-2D similar a la anterior pero usa dos bloques y no solo uno; finalmente la opción *d* muestra una distribución 2D-2D con dos filas y dos columnas de bloques (2D) y cada bloque tiene una distribución 2D de hilos.

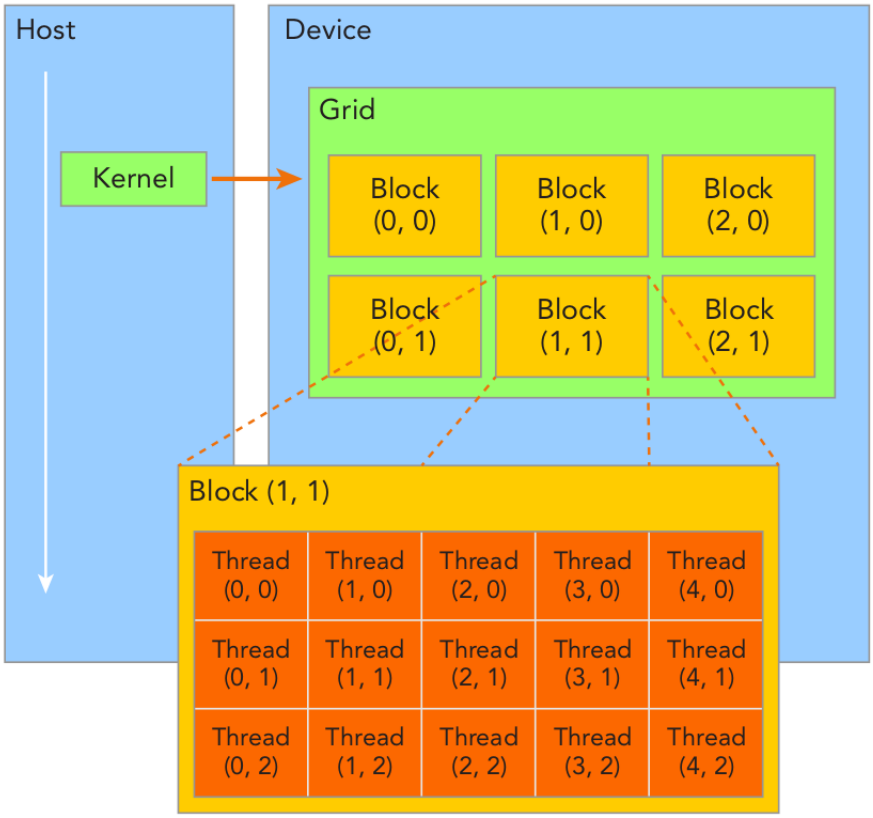

Figura 15. Esquema de diseño de hilos para implementar paralelismo en tarjetas GPU Nvidia. Tomado de Cheng et al. (2014).

Del esquema presentado en la figura 16, se puede entender porque que la ejecución del kernel en GPU puede tener varias distribuciones que llevan a los mismos resultados, sin embargo, la importancia de seleccionar el diseño apropiado radica en el uso de los recursos de la GPU y el desempeño obtenido por la GPU cuando se ejecutan los hilos ya que no vale la pena obtener solo resultados con una distribución pero tomando el doble o triple del tiempo de ejecución que al utilizar otra distribución o gastando tres o mas veces el escaso recurso de memoria.

Para poner en contexto, si la implementación heterogénea de FWI toma 3 horas para obtener un resultado adecuado con un diseño de hilos específico, no es lógico utilizar otra distribución que

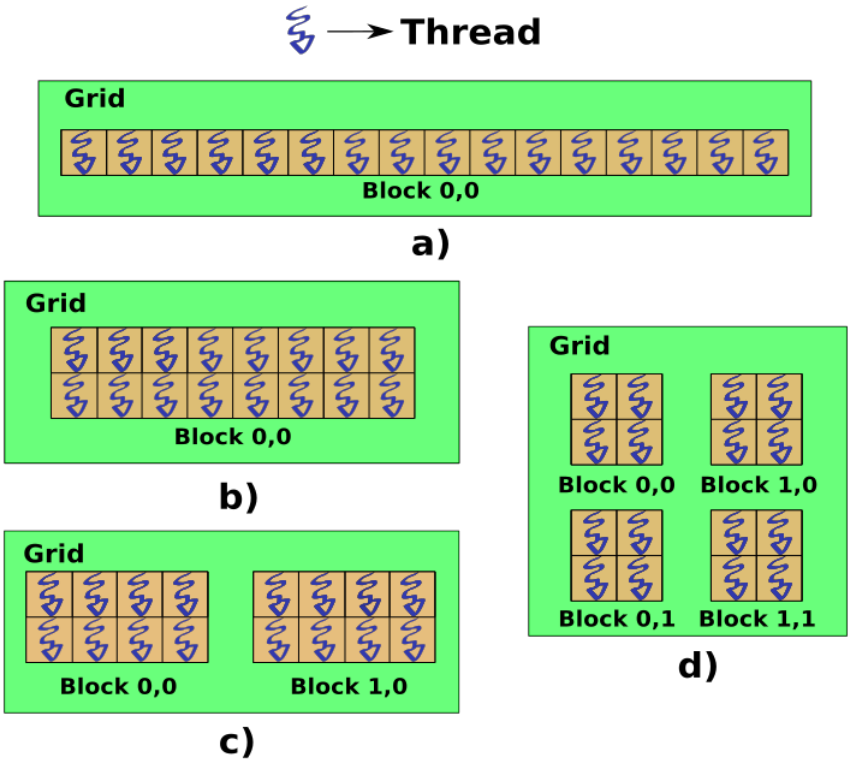

Figura 16. Ejemplos de diseño de hilos para un kernel que se ejecuta sobre 16 hilos: a) diseño 1D-1D , b) diseño 1D-2D , c) diseño 1D-2D , d) diseño 2D-2D.

toma 6 horas en obtener los mismos resultados; sin embargo, la selección del diseño apropiado no es una tarea sencilla ya que no existe un procedimiento sistemático sino empírico y que depende del tipo de aplicación. A pesar de la dificultad en la determinación del diseño, Nvidia ha elaborado unos documentos NVIDIA (2020b,a) con recomendaciones a la hora de programar paralelismo en su hardware, estos deben considerarse fuente primaria de consulta al implementar algoritmos en GPU.

5. Estrategia computacional para la implementación de FWI 2D elástica

Considerando el fundamento matemático ya descrito y las herramientas computacionales, es posible ahora proceder con la implementación numérica de la FWI elástica; para dar una mejor descripción del procedimiento la figura 17 muestra el diagrama de flujo general que contiene las principales etapas involucradas.

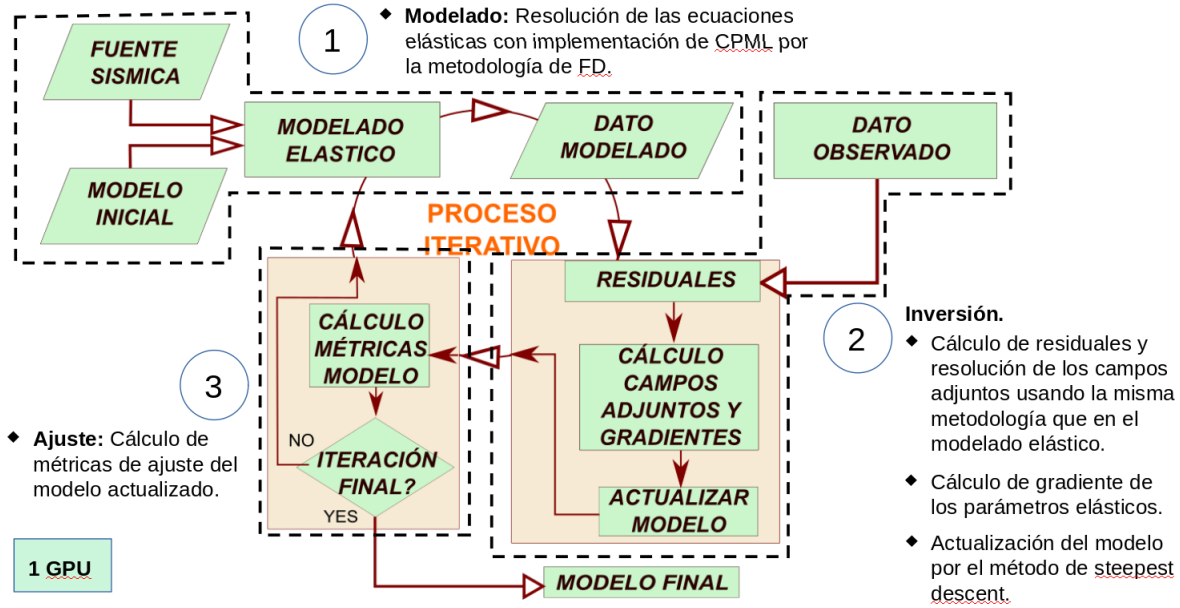

Figura 17. Diagrama de flujo general para la implementación de la FWI elástica en GPU.

En términos generales la implementación de la FWI se puede dividir en 3 grandes etapas, la primera denominada *modelado* corresponde a la ejecución del modelado de onda elástico que involucra la selección de la fuente sísmica a utilizar, en nuestra caso esta dada por la ondícula Ricker presentada en la ecuación 6 y el modelo inicial sobre el cual se propagará la onda; el siguiente paso involucra la solución de las ecuaciones de propagación incluyendo la modificación

para integrar las zonas CPML que se presentanen las ecuaciones 8 , 9 , 10 , 11 y 12 a través del método de diferencias finitas que se rige por las expresiones 21 , 22 y 23 ; los resultados del modelado de onda incluyen la generación del sismograma modelado el cual se corresponde con el *dato modelado* que será utilizado en el cálculo de los gradientes.

En la segunda etapa denominada *inversión* se realiza inicialmente la comparación entre el *dato modelado* y el *dato observado* que lleva a la generación de los *residuales*, estos valores de los residuales pasan a ser las fuentes utilizadas en la retro propagación a la hora de resolver las ecuaciones adjuntas dadas por las expresiones 28 , 29 , 30 , 31 y 32 . Es importante mencionar quue la solución de las ecuaciones adjuntas involucra nuevamente la modificación para añadir la implementación de las zonas CPML de forma similar a lo hecho en el modelado; la solución de este sistema de ecuaciones se realiza mediante el método de diferencias finitas manteniendo las mismas características utilizadas en el modelado.

Al final de esta etapa se realiza el cálculo de los gradientes de parámetros elásticos con la información proveniente del modelado y la retro propagación siguiendo las ecuaciones 33 , 34 y 35 , estos gradientes son el insumo para realizar la actualización del modelo inicial por el método de gradiente descendente con la aplicación de la ecuación 27 individualmente por cada parámetro.

La útima etapa denominada *ajuste* involucra la evaluación del modelo actualizado en cada iteración para analizar si el modelo resultante va mejorando comparando cada vez con el modelo inicial. Esto puede ser realizado utilizando diferentes métricas en la literatura. Sin embargo, se suele utilizar la evolución de la función de costo y la evolución en el error cuadrático medio (MSE), el principal objetivo de esta etapa es evaluar si el modelo elástico obtenido después de aplicar FWI

lleva a resultados apropiados y realistas.

Es necesario realizar una mención final relacionada con el lugar donde las instrucciones se ejecutan, la FWI es un proceso altamente paralelizable por tanto la idea es ejecutar la mayoría de las etapas en GPU como se resalta con el color verde en la figura 17, sin embargo, se debe aclarar que debido a que el paradigma utilizado es heterogéneo siempre tendrá que realizar algunas intervenciones de la CPU que pueden ser en controlar el flujo del programa, guardado de variables de resultados u otras. A pesar de esto, las tareas de alta exigencia computacional de procesamiento incluidas en el código se programarán para ser ejecutadas en GPU.

5.1. Consideraciones para el diseño de hilos en GPU para la FWI elástica

Las GPU Nvidia contienen SM como el componente principal para lanzar los hilos y bloques, el SM programa los recursos necesarios para ejecutar los kernels que contienen el código a ser paralelizado a través de la aplicación de los diseños de hilos Cheng et al. (2014); NVIDIA (2020b). En el caso de la inversión de onda completa elástica el código paralelo incluye el modelado, la retro propagación, el cálculo de los gradientes. Una vez la cuadrícula de discretización es creada, es posible asignar los recursos de memoria de la GPU (en forma matricial) y diseñar las distribuciones de hilos para correr los kernel y ejecutar la FWI.

Cuando se implementa una discretización que genera una matriz normalmente la primera aproximación considera utilizar un hilo para realizar operaciones por cada espacio de memoria asignado de esa matriz ya que es la forma mas simple y un opción sencilla, esta distribución lleva a un diseño 1D donde existe un gran bloque que contiene todos los hilos necesarios para cubrir el espacio del modelo lo que lleva a un número total de hilos por bloque de $N_x \times N_z$. A pesar de que

este diseño es la primera opción por su simplicidad, en términos prácticos no es una buena opción ya que la arquitectura de hardware limita el tamaño de los bloques a un máximo de 1024 hilos lo cual limita directamente el tamaño del modelo que se puede procesar de esta forma.

Otro diseño considera la asignación de los hilos de procesamiento que se correspondan con N_x espacios de memoria y el número de bloques se fija para ser el valor de N_z que representa el número de filas en la matriz, esta configuración mantiene la forma 1D del ejemplo anterior pero en este caso el número de bloques utilizados se incrementa, este diseño se puede detallar en la figura 18 a. Este diseño mejora el anterior al permitir trabajar con modelos de mayores tamaños aunque aun existe una limitante por el tamaño del bloque ya que no puede procesar una matriz de mas de 1024 filas.

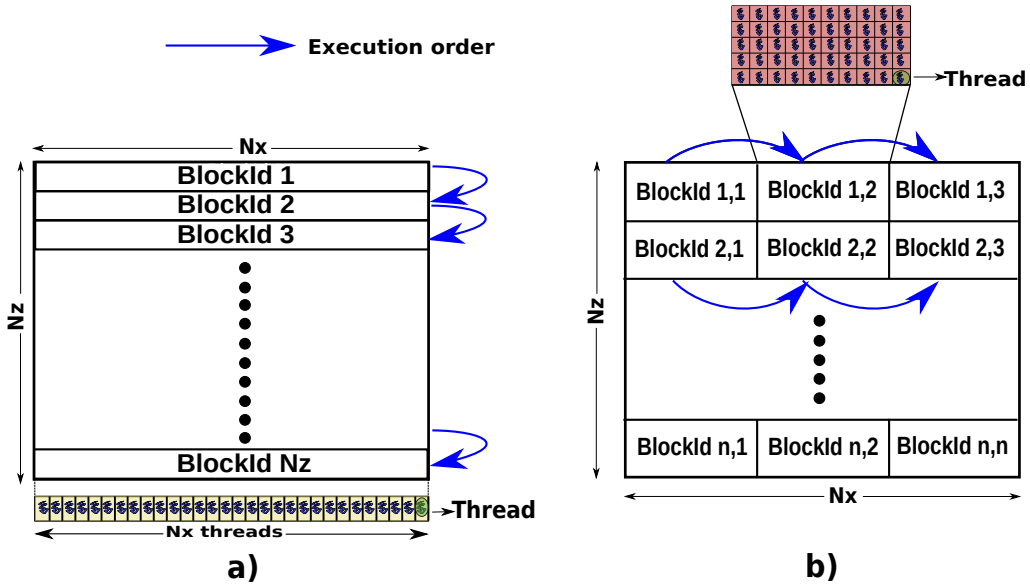

Figura 18. Opciones de diseño de hilos para la implementación de FWI elástica: a) diseño 1D, b) diseño 2D.

Adicional a lo mencionado anteriormente sobre la asignación de recursos en el hardware

Nvidia cuando se lanzan las distribuciones de hilos, es importante mencionar que la arquitectura de GPU considera internamente la división de los hilos por *warps*. cuando se lanza un kernel todos los hilos asignados para ejecutarlo se ajustan al tamaño del warp y posteriormente algunos de estos se ejecutan de forma simultánea.

La cantidad de warps disponible depende de la distribución de registros y memoria compartida realizada cuando un diseño de hilos se establece, sin embargo, esta progamación de recursos no es directamente controlada por el programador ya que esta tarea la realiza directamente la GPU. Por este motivo, un diseño de hilos apropiado es también importante para la lograr un mejor uso de los recursos de la GPU, sin embargo, no existen instrucciones específicas sobre como generar estos diseños y por tanto el procedimiento general se vuelve empírico que depende principalmente del tipo de aplicación a ejecutar y la disponibilidad de hardware en determinado instante.

A pesar de lo mencionado, Nvidia ha entregado algunas sugerencias NVIDIA (2020a) que pueden ayudar en la generación de diseños apropiados; ellos recomiendan mantener el número de hilos por bloque en un valor que sea múltiplo del tamaño de warp, evitar utilizar bloques de tamaños pequeños y seleccionar la cantidad de bloques que sea un número bastante mayor a la cantidad de SM de la GPU. Al tomar estas consideraciones en cuenta se puede generar un nuevo diseño de hilos como el que se presenta en la figura 18 b.

La distribución de hilos se logra al dividir la matriz de discretización en pequeñas matrices 2D de tamaño específico para cubrir el espacio de la matrix mas grande, estos pequeños cuadros corresponderán al número de bloques en el diseño y cada uno de estos bloques trabajará en una distribución de hilos 2D. De esta forma es posible trabajar con mayor cantidad de bloques mientras

se mantienen el número de hilos como un valor múltiplo del tamaño de un warp, al mismo tiempo este diseño permite tratar con modelos de dimensiones superiores sin tener que implementar ningún cambio mayor.

Existen muchas posibilidades para seleccionar el diseño 2D apropiado para determinado modelo, una sugerencia útil puede ser el ejecutar kernels con tamaño de bloque de 128 o 256 hilos en una disposición 2D dando una configuración de 16x16 hilos por ejemplo, algunas pruebas fueron realizadas para el caso del modelado elástico utilizando diseños de hilos 1D y 2D y midiendo el desempeño en términos de tiempo de ejecución Chanaga et al. (2020a); los resultados muestran que la aplicación del diseño 2D puede reducir el tiempo de ejecución al ejecutar el modelado comparado con una implementación usando un diseño 1D y esta mejora se ve incrementada cuando se consideran modelos de mayores dimensiones y con mayores tiempos de simulación.

5.2. Implementación de CPML en GPU, estrategia común y de memoria reducida

Cuando se implementa modelado y retro propagación numérica es crucial la definición de las zonas absorbentes para el procesamiento CPML, estas áreas de grosor fijo y extensión variable se muestran en la figura 19 a. En la imágen, se puede claramente diferenciar dos grandes áreas llamadas *Zona Regular* y *Zona CPML*, por la primera zona, la onda propagada viaja inalterada mientras en la última la energía de la onda es reducida a medida que se profundiza en el ingreso sobre esa área.

Cada zona CPML tiene un grosor específico de L_x o L_z que se extiende hasta cubrir totalmente la dimensión correspondiente N_z o N_x, cuando se implementa sobre GPU estas zonas deben tener su propio espacio de memoria asociado con el uso de variables auxiliares diferentes a la me-

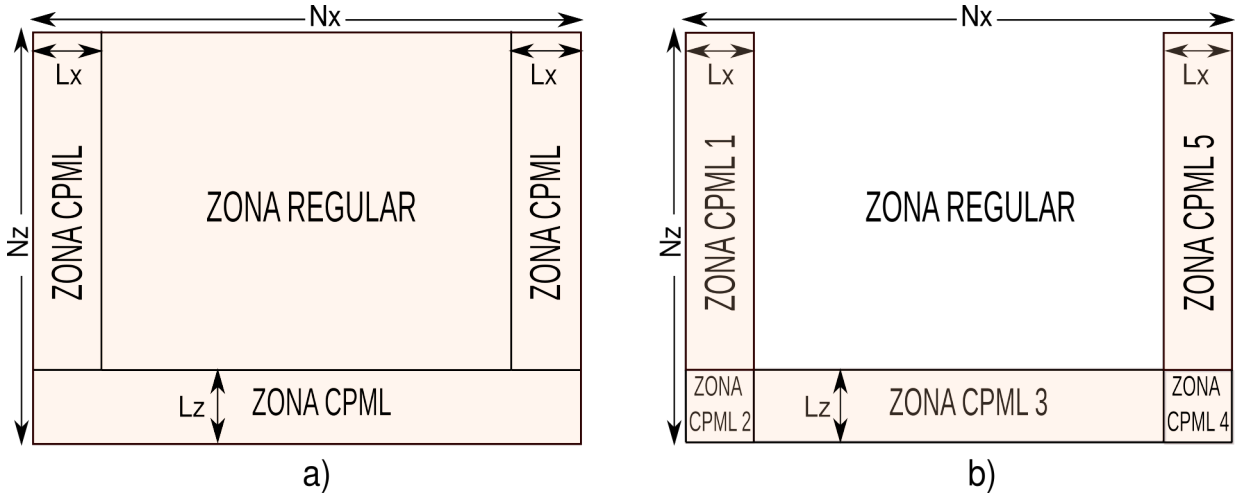

Figura 19. Distribución de las zonas CPML asignación de recursos: a) estrategia común para la asignación de CPML, b) Estrategia de memoria reducida CPML.

moria asignada para realizar la actualización de los campos de velocidad y esfuerzos, sobre estos espacios de memoria un nuevo conjunto de instrucciones son ejecutadas con cálculos específicos del método CPML.

Es común encontrar que las implementaciones de CPML en la FWI utilizan espacios de memoria de tamaños $N_x \times N_z$ bytes, el cual corresponde al tamaño de un modelo completo adicional al espacio del modelo ya asignado en el cálculo de actualización de los campos, se hace de esta forma buscando mantener la implementación del programa lo mas sencilla posible pero sacrificando recursos de memoria de GPU valiosos.

Cuando se usa el esquema común la cantidad de recursos de GPU invertidos en operaciones es mayor ya que en cada iteración un kernel CPML cubre completamente el espacio de memoria del modelo asignado (coloreado en la figura 19 a) incluyendo la zona regular donde se supone que no debe realizar ninguna instrucción CPML. Al cubrir este espacio también ocurren accesos a índices

de memoria, saltos sobre índices e instrucciones internas que llevan a un uso ineficiente de los hilos y la memoria de la GPU lo cual impacta en el tiempo de ejecución, además, este comportamiento ineficiente puede empeorarse si se consideran modelos de dimensiones mas grandes.

Otra alternativa para solucionar el tema del procesamiento de las zonas CPML considera la creación de 5 zonas CPML con una asignación reducida de memoria para mantener un mejor control en el recurso de memoria y en la ejecución del código, además los kernels para operaciones CPML evitan el acceso a la zona regular, esta implementación es mostrada en la figura 19 b. A diferencia de la implementación común la nueva propuesta considera 2 asignaciones de memoria independientes de tamaño $L_x \times (N_z - L_z)$ bytes, otras 2 de $L_x \times L_z$ bytes, finalmente 1 área of $(N_x - 2L_x) \times L_z$ bytes; utilizar este esquema asegura que la memoria total utilizada por la GPU debido a las CPML se reduce y esta tendencia es especialmente notable en los casos donde los tamaños de los modelos son mas grandes debido a que las fórmulas de asignación de memoria dependen solo de 1 dimensión que proviene del modelo mientras se mantienen constantes las otras que en la mayoría de los casos es un valor pequeño L_x or L_z, por el contrario en la implementación común la asignación de memoria depende de las 2 dimensiones del modelo.

Es importante mencionar que la cantidad de recursos de GPU ahorrados en la estrategia de memoria reducida de CPML se ve disminuido cuando las dimensiones del modelo decrecen llevando en algún punto a un virtual empate en el desempeño para ambas implementaciones de CPML, sin embargo, las aplicaciones prácticas de la FWI elástica suelen trabajar sobre modelos de grandes dimensiones y por tanto es muy probable tener la necesidad de utilizar una estrategia de manejo eficiente de memoria para el cálculo de las CPML como la prsentada aquí.

La dificultad en la implementación de la estrategia de memoria reducida para el cálculo de las CPML radica principalmente en la necesidad de utilizar dos índices diferentes para acceder a memoria y ejecutar las instrucciones sobre los campos, además de lograr la conexión apropiada entre esos índices, por otro lado la implementación común solo trabaja con un índice lo que simplifica de gran manera la forma de lograr la interacción entre los distintos campos y de estos con las zonas CPML pero con el problema del uso ineficiente de los recursos.

5.3. Administración de memoria CPU-GPU para almacenamiento del cubo de propagación

De acuerdo a la guía de Nvidia de mejores prácticas NVIDIA (2020a), la optimización en el manejo de la memoria de la GPU es el aspecto clave a desarrollar cuando se busca mejorar el desempeño de un algoritmo, esto es particularmente cierto en el modelado elástico donde se esperan altos niveles de mejora especialmente en tiempo de ejecución y consumo de memoria debido a que este procedimiento hace parte de metodologías mas complejas como la FWI Brittan et al. (2013); Virieux et al. (2017) y migración Menke (2018).

Programar en GPU involucra la aplicación de computación heterogénea Arora (2012) donde una CPU actúa como el controlador del flujo de ejecución del programa y la GPU realiza las operaciones de alta exigencia computacional; cada dispositivo posee su propio espacio de memoria y un algoritmo bajo este paradigma necesariamente tendrá que compartir información entre dichos espacios.

La arquitectura de GPU Nvidia sustenta la ejecución de hilos de procesamiento sobre una jerarquía de memoria la cual incluye varios tipos de memoria como los registros y la memoria compartida que son las rápidas pero es un recurso muy limitado, mientras tanto la memoria global

(VRAM) es mas lenta pero se tiene una mayor disponibilidad para el uso lo que la hace ideal para tratar con el gran tamaño de los cubos de propagación de los campos utilizados en la FWI.

La CPU también cuenta con espacios de memoria para almacenar información que incluye la memoria RAM y los disco rígidos por mencionar los mas conocidos; estos espacios son en la mayoría de casos de un mayor tamaño que las opciones disponibles en GPU lo que los hace mas adecuados para el almacenamiento de los cubos de propagación y ya que que se está trabajando bajo la programación heterogénea implica que los datos se pueden transferir de un espacio a otro.

Al implementar FWI es importante mantener la información de los campos de velocidad y esfuerzo actualizados y posteriormente generar el cubo de propagación con estos campos, para lograr esto la figura 20 presenta un opción de almacenamiento donde la GPU realiza la actualización del campo y transfiere a la memoria de sistema de la CPU en cada iteración. En esta opción, el modelado se ejecuta por los kernels en GPU y se actualiza el campo en cada iteración, una vez el campo es actualizado una transferencia de datos lleva el campo de la memoria global a CPU sobre el bus PCI-e; cuando el campo llega la memoria de sistema en CPU es almacenado y la ejecución de la siguiente iteración para actualizar el campo es permitida.

Esta implementación explota la mayor cantidad de espacio disponible en la memoria RAM de la CPU donde puede llegar hasta unos 32 GB en estaciones de trabajo y a varios cientos de Gigabytes cuando se consideran los clusteres, mientras que la memoria global de la GPU suele rondar sobre los 4 u 8 Gigabytes en las tarjetas de consumo y desde los 16 GB a los 32 GB en las mas costosas utilizadas en servidores.

La principal desventaja de esta estrategia se asocia con el ancho de banda reducido dispo-

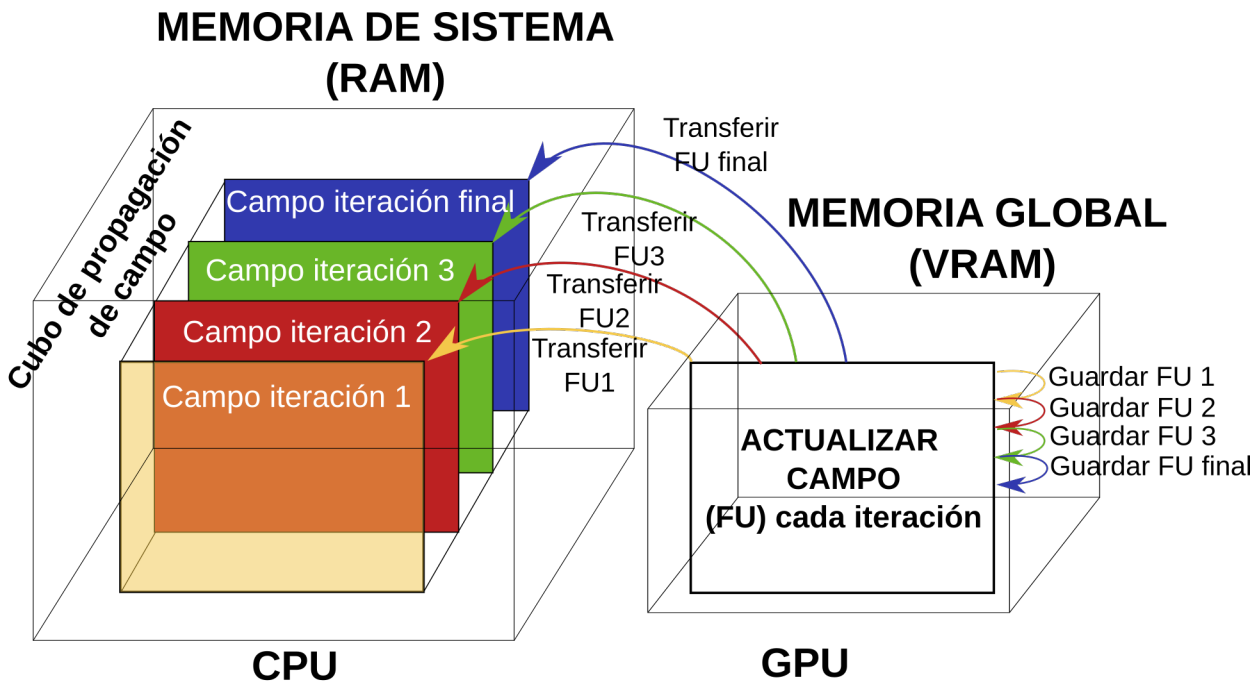

Figura 20. Almacenamiento del cubo de propagación en la memoria de sistema (RAM) de la CPU cada vez que la GPU actualiza el campo.

nible en el bus PCI-e x16 llegando a un máximo de 16 GB/s mientras el ancho de banda de la memoria VRAM puede llegar a un máximo de 128 GB/s (en la tarjeta Nvidia Gtx 1650) lo que permite un rata de transferencia superior. Además del ancho de banda, el *overhead* debido a las transferencias GPU-CPU pueden impactar el tiempo de ejecución del modelado y a la vez de la FWI.

Otra estrategia para guardar el campo se muestra en la figura 21, en este caso la GPU actualiza el campo pero lo mantiene en la memoria global y lo almacena ahí mismo de tal forma que se va creando el cubo, una vez el cubo esta completa una transferencia de datos hacia la CPU toma lugar si es necesario. Esta estrategia resuelve la mayoría de las desventajas del esquema previo al explotar el mayor ancho de banda en las transferencias internas de la GPU acelerando la

ejecución de los kernels y la construcción del cubo. De la misma forma la GPU realiza solo una gran transferencia de datos a CPU lo que disminuye el impacto en tiempo de ejecución debido al overhead.

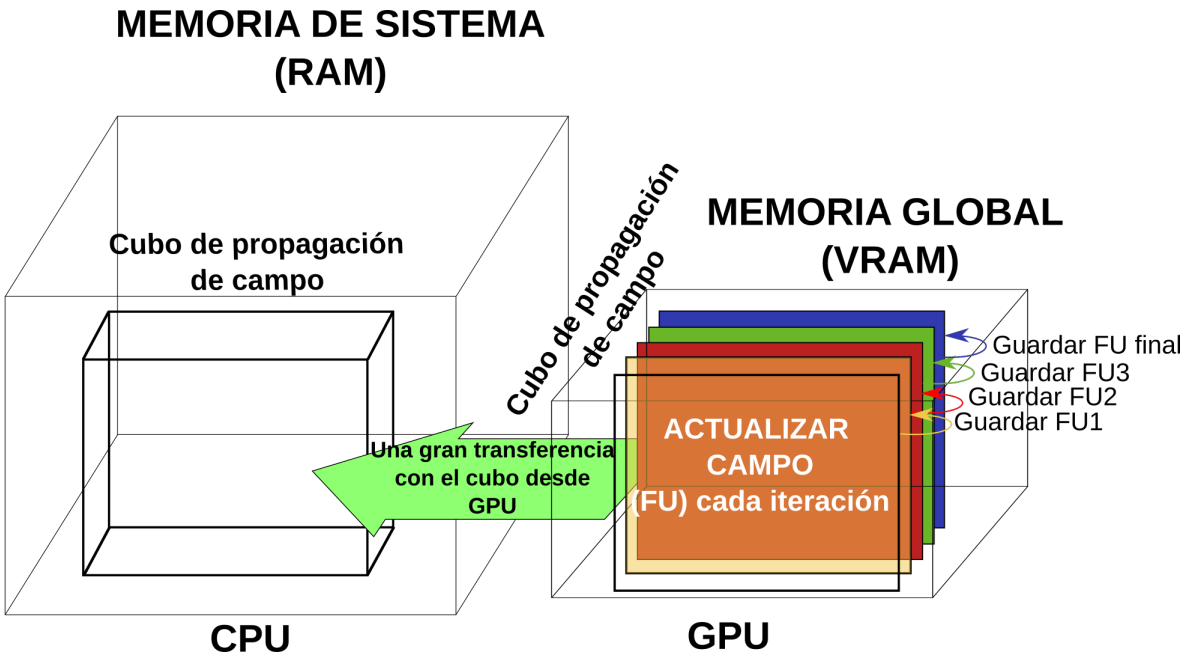

Figura 21. Almacenamiento del cubo de propagación en la memoria global de la GPU (VRAM) cada vez que la GPU actualiza un campo.

Cual estrategia utilizar depende del monto de memoria necesario para la implementación de FWI, esto incluye memoria para almacenar los campos de velocidad y esfuerzos, los adjuntos, el almacenamiento de los cubos de propagación y otras consideraciones. Sin embargo, es claro que la FWI con modelos de gran tamaño debería utilizar la capacidad de memoria que se encuentra en la CPU con la desventaja del bajo ancho de banda en las transferencias de dato y los mas lentos módulos de memoria RAM comparado con los utilizado en las GPU. A pesar de ello, si es posible contar con tarjetas GPU de última tecnología o clusteres de GPU, la desventaja en cuanto espacio

disponible se ve reducida y la balanza se inclina hacia el uso de la memoria de la GPU debido a sus características de alto desempeño.

Algunas pruebas fueron realizadas para medir el impacto de implementar tácticas computacionales como el diseño de hilos, la programación común o con memoria reducida de las CPML y el almacenamiento del cubo de propagación anteriormente mencionado; al ejecutar modelado elástico con el uso combinado de diseño 2D de hilos, CPML de memoria reducida y almacenamiento del cubo en la memoria de la GPU contra una implementación usando diseño 1D de hilos, estrategia común de cálculo de CPML y almacenamiento del cubo en CPU Chanaga et al. (2020b); Los resultados mostraron grandes mejoras en tiempo de ejecución y consumo de memoria de la primera estrategia respecto de la segunda debido principalmente al uso mas eficiente de los recursos de la GPU.

Los resultados de esas pruebas inclinan la balanza hacia la utilización de la primera estrategia en el desarrollo de la FWI elástica.

5.4. Estrategia computacional para la implementación de FWI 2D elástica en GPU usando el paradigma heterogéneo

Hasta este punto, se ha presentado toda la teoría necesaria para la implementación de la FWI, se introdujo el paradigma de programación heterogénea y la nueva área de investigación HPC; también se mostró la importancia de las GPU y su uso para resolver los nuevos problemas con alto costo computacional en término de consumo de recursos y tiempo de ejecución. Se explicó brevemente la arquitectura de la GPU y los fundamentos para la implementación de paralelismo para resolver los sistemas de ecuaciones diferenciales discretizadas.

Al inicio de la sección se presentó un diagrama flujo para la implementación de FWI elástica detallando las principales actividades a tener en cuenta, de igual forma, se presentó algunas consideraciones con la finalidad de obtener mejor desempeño cuando se implementa el código de FWI en un sistema heterogéneo enfocándose en recomendaciones de administración; ahora, con toda esta base teórica y computacional es posible introducir la estrategia de implementación de la FWI 2D elástica.

En la figura 22 , se muestra la estrategia de implementación de FWI 2D elástica, se puede ver como una adaptación del diagrama de flujo general en término de espacios computacionales de CPU y GPU, también se detalla las estructuras de control y se convierten las principales actividades mostradas en la figura 17 en instrucciones computacionales; de esta forma se puede tener mayor claridad del porqué utilizar programación heterogénea.

Para iniciar es necesario configurar los parámetros de la FWI, en esta tarea se incluye la lectura del modelo inicial que corresponde a los tres parámetros ρ, V_s y V_p, el valor de los parámetros de discretización Δt y Δh dando cumplimiento al criterio de estabilidad numérica, los parámetros para la ondícula Ricker que en general es la frecuencia fq, las dimensiones del modelo y el número de fuentes a ser utilizadas en la inversión incluyendo sus posiciones dentro del modelo.

Junto con estos parámetros se debe definir los parámetros CPML incluyendo el grosor L_x y L_z y todas las constantes asociadas con los cálculos de las variables auxiliares; en este punto se especifica también los punteros que serán utilizados tanto en CPU como en GPU para el almacenamiento y cálculos intermedios así como los resultados parciales. Aquí sucede la asignación de memoria en GPU y la transferencia desde la CPU a GPU del modelo inicial, el dato verdadero y las

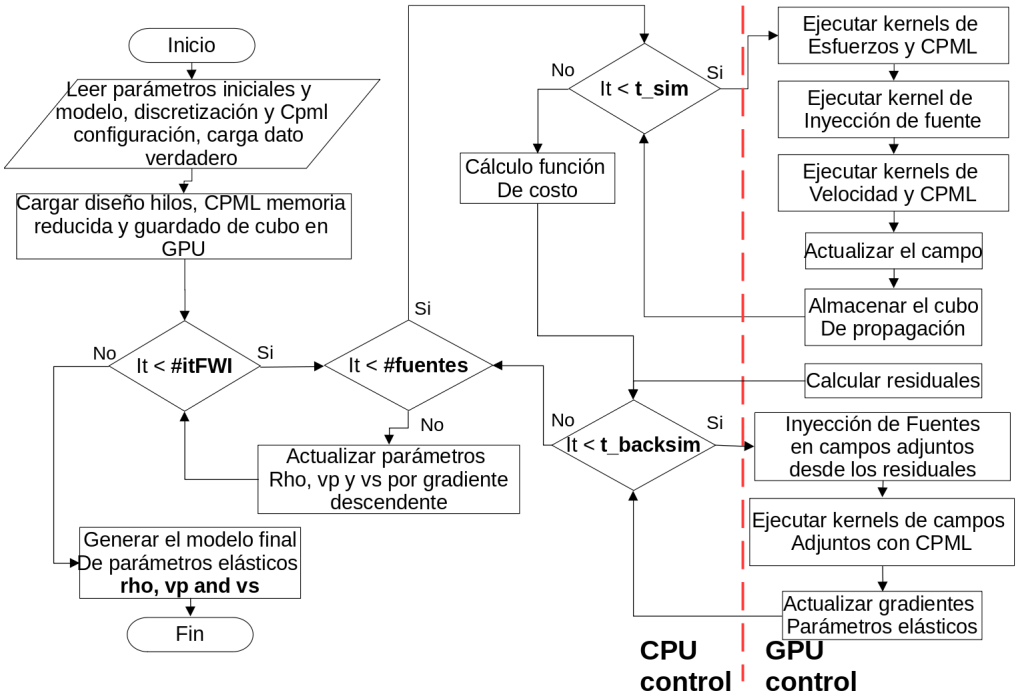

Figura 22. Diagrama de flujo para la estrategia de implementación computacional de la FWI 2D elástica utilizando el paradigma heterogéneo.

variables intermedias para ejecutar operacions en GPU ya que este hardware por si solo no puede realizar esas operaciones, hace parte de las tareas de control de la CPU.

Una vez finalizado el proceso de cargue de información, el algoritmo procede con la definición del diseño de hilos utilizando en este caso el esquema 2D con un tamaño de bloque de 16x16 y una cuadrícula 2D resultante de dividir las dimensiones del modelo discretizado N_x y N_z sobre el número de hilos por bloque es 256; con la información del diseño y los parámetros CPML se definen los índices para operar en los kernels bajo la estrategia de cálculo por memoria reducida de CPML y también los índices en la zona regular; finalmente como se mencionó el cubo de propagación será almacenado en memoria global (VRAM) en GPU a pesar de su mas pequeño tamaño

comparado con la memoria RAM pero se tomará ventaja de su mayor ancho de banda y frecuencia de trabajo.

Con la configuración ya establecida, el próximo paso es la definición de la cantidad de iteraciones FWI, este valor se suele elegir con relación a la evolución de la función de costo ya que se espera que a mayor cantidad de iteraciones la función de costo disminuya y mejore la reconstrucción de los modelos de parámetros elásticos; después de esto, el procedimiento de FWI continua considerando cada sismograma verdadero disponible y genera su correspondiente sismograma modelado al utilizar el mismo número de fuentes que las utilizadas para obtener cada sismograma verdadero; esto se debe realizar de esta forma para lograr la comparación uno a uno entre los sismogramas y poder generar los residuales.

Avanzando en el procedimiento, la próxima etapa involucra la ejecución del modelado elástico al iterar a través del tiempo de simulación discretizado *t_sim*; en cada iteración los kernel de cálculo de los esfuerzos y las velocidades se ejecutan en GPU resolviendo el sistema de ecuaciones de propagación modificado al incluir la implementación de las CPML y generando también el cubo de propagación; es importante mencionar que la primera propagación de onda elástica sucede sobre el modelo inicial de parámetros elásticos ρ, V_s y V_p que fueron previamente cargados en GPU en las primeras etapas del proceso.

Una vez el cubo de propagación se completa, el algoritmo continua con el cálculo de los residuales al comparar el sismograma verdadero con el sismograma generado en el modelado que ya finalizó, igualmente con esos residuales es posible proceder con el cálculo de la función de costo para evaluar la evolución de la FWI en esta iteración.

Con la información de residuales se puede proceder a ejecutar la retro propagación; el procedimiento es similar al realizado para el modelado, sin embargo, las fuentes de los campos adjuntos corresponden a los residuales y el tiempo discreto avanza en la dirección temporal contraria a la asumida en el modelado iniciando la primera operación en el tiempo final y retrocediendo en el tiempo de simulación. En cada iteración de la retro propagación se realiza una actualización de los gradientes de parámetros elásticos hasta que el total de tiempo de simulación se termina.

El modelado y la retro propagación se ejecutan por cada sismograma hasta que se obtiene un gradiente final para cada parámetro elástico, con esos gradientes es posible proceder con la actualización del modelo inicial hacia un nuevo modelo actualizado utilizando el método de gradiente descendente. En este punto finaliza la primera iteración y se reinicia el procedimiento para la siguiente iteración FWI.

Cuando la última iteración FWI termina el procedimiento se detiene y se genera el modelo final de los parámetros elásticos con la imágen de la composición del subsuelo de la zona de estudio; en este punto se deben realizar algunos comentarios para clarificar el costo computacional de la implementación de la FWI.

Se debe enfatizar que el modelado y la retro propagación ocurren por cada fuente considerada y por tanto depende de la cantidad de sismogramas verdaderos con los que se cuente, por ejemplo si hay 7 sismogramas verdaderos el proceso completo en GPU debe repetirse 7 veces lo que impacta directamente en el tiempo de ejecución y la disponibilidad de recursos de hardware; sin embargo, la importancia de utilizar varios sismogramas verdaderos de fuentes ubicadas en distintas posiciones radica en que distintos ángulos de visión de una misma zona en una adquisi-

ción de campo pueden ayudar en la mejor definición de los gradientes y estos a su vez permitiran mejorar la reconstrucción de los modelos de parámetros elásticos.

A pesar de esta ventaja en la mejora de los gradientes, este proceso puede aumentar el costo computacional de la FWI si la implementación no se enfoca de forma correcta, por ejemplo, si una iteración FWI para un sismograma toma 30 segundos y si se intenta ejecutar la inversión para un total de 100 o mas sismogramas verdaderos, el tiempo total invertido en la ejecución será de 3000 segundos o 50 minutos que puede ser aceptable, sin embargo, si se considera ejecutar 20 iteraciones FWI el costo en tiempo aumenta hasta llegar a 1000 minutos, lo que ya puede llevar a pensar que el tiempo invertido es inaceptable.

Debido a estos inconvenientes de costo computacional en la implementación de la FWI es que los investigadores se mantienen en constante búsqueda y desarrollo de nueva estrategias desde diferentes frentes para mejorar el tiempo y los recursos invertidos en la inversión, estas estrategias exploran desde nuevas formulaciones matemáticas, reconstrución de campos, técnicas de optmimización, el uso de *tensor cores* para cálculos matriciales entre otros; sin embargo, se debe aclarar que estas estrategias estan fuera del alcance de este trabajo y no serán consideradas.

6. Resultados

Con la estrategia de implementaón presentada es posible ahora hablar sobre los resultados computacionales, pero antes se debe mencionar el equipo utilizado para la realización de las pruebas. Se cuenta con una estación de trabajo que consiste de un procesador AMD Ryzen 5 3550H CPU con un valor de memoria RAM de 16GB, la estación cuenta con una tarjeta GPU Nvidia Gtx 1650 con un valor total de memoria (VRAM) de 4 GB Video, el sistema operativo sobre el que se programó es Debian Linux version 10 (Buster) y los lenguajes de programación utilizados fueron CUDA-C junto con C para un sistema heterogéneo.

Las mediciones de ajuste para cada prueba incluyen la evolución de la función de costo calculada como se muestra en la ecuación 36, esta métrica está directamente asociada con los residuales entre el dato modelado d_{modR} y el dato verdadero d_{trueR} en cada iteración FWI y es calculada para cada receptor R en el sismograma.

$$CF = \frac{1}{2}\sum_{R}^{N_R} ||d_{modR} - d_{trueR}||^2. \tag{36}$$

De igual forma, como se va a trabajar sobre información sintética para el dato verdadero, se cuenta con el modelo verdadero de parámetros elásticos desde el principio y esto permite poder comparar el modelo resultado de cada actualización al aplicar FWI con la respuesta esperada. Esto facilita la utilización de otra métrica de ajuste conocida como el error medio cuadrático (MSE) que se rige por la ecuación 37 ; aquí la expresión W es el modelo verdadero y $\tilde{W}$ es el modelo

actualizado obtenido después de cada iteración FWI,

$$MSE(W,\tilde{W}) = \frac{1}{N_x N_z}\sqrt{\sum_{i=0}^{Nx-1}\sum_{z=0}^{N_z-1}\left(W-\tilde{W}\right)^2}. \tag{37}$$

El banco de pruebas consta de dos pruebas, la primera considera un modelo elástico sintético de creación propia, la segunda prueba considera un modelo obtenido de la literatura como es el caso del overthrust 2D. Es importante recordar en este punto que todas las pruebas fueron realizadas sobre datos sintéticos que significa que no existe datos reales provenientes de adquisiciones, por tal motivo los sismogramas verdaderos se generarán a partir de la ejecución previa de un modelado elástico sobre el modelo real de parámetros elásticos para generar los sismogramas que serán considerados los verdaderos dentro del procedimiento de FWI.

6.1. Modelo CPS

6.1.1. Medidas de ajuste para la FWI. La primera prueba involucra la definión de un modelo de parámetros elásticos de propia autoría, se decidió por tanto trabajar con una imágen con el logo CPS y se adaptó para construir los parámetros elásticos ρ, μ y λ con esta forma, sin embargo, para facilitar la interpretación de los resultados se utilizará la forma ρ, v_p y v_s y los resultados de la FWI serán presentados también de esta forma.

En la figura 23 , se muestra el modelo verdadero al que se quiere llegar luego de aplicar la FWI sobre un modelo inicial, este modelo tiene unas dimensiones de $N_x = 214$ por $N_z = 115$ puntos, el paso de discretización espacial es $\Delta h = 2.5$ [m] y el paso temporal es $\Delta t = 0.75$ [ms], estos datos nos llevan a un modelo de 535 [m] por 287.5 [m]. Con esta selección de parámetros se

da cumplimiento a los criterios de estabilidad nuérica y dispersión, la inversión se aplica para un total de 7 fuentes ubicadas a 7.5 [m] (3 puntos) de profundidad, ubicando la primera a 87.5 [m] (35 puntos) en la dirección *offset* y las posteriores separadas cada 60 [m] (24 puntos); la frecuencia central de la ondícula Ricker es de 25 [Hz].

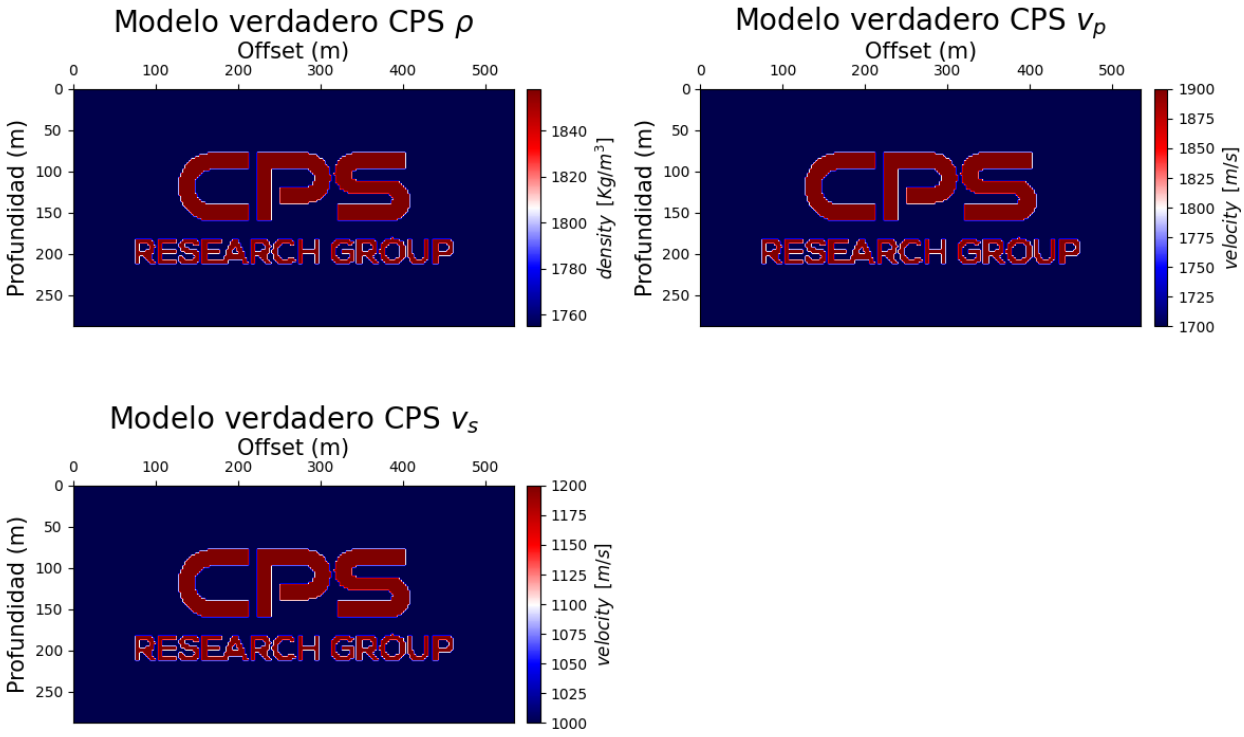

Figura 23. Modelo sintético verdadero CPS para evaluar la FWI programada.

Los valores de los parámetros elásticos varían de 1755 a 1859 [Kg/m^3] para la densidad ρ, 1700 a 1900 [m/s] para la velocidad v_p y 1000 to 1200 [m/s] para la velocidad v_s; la inversión se ejecutará sobre unas 2000 muestras de tiempo generadas de un tiempo de simulación de 1.5 [s].

Los sismogramas modelados fueron obtenidos por un conjunto de receptores ubicados a 10

[m] (4 puntos) de profundidad y separados entre ellos cada 2.5 [m] en la dirección *offset*, estos receptores actúan como geófonos midiendo las velocidades v_x y vz ya que ambas se tienen en cuenta en la inversión; por tanto se deben procesar 2 sismogramas para cada fuente utilizada.

El método de gradiente descendente utilizado en la FWI necesita de la definición de un parámetro α el cual indica el factor de escala para el gradiente en el proceso de actualización de cada parámetro elástico; a pesar de que existen algunas formulaciones para estimar este valor puede no llegar a ser suficiente preciso y por tanto la selección suele realizarse de forma empírica a prueba y error, en esta implementación la selección de α_λ, α_μ y α_ρ fue realizada partiendo de un valor inicial y verificando que existiera una disminución en la función de costo durante todo el tiempo de simulación de la FWI.

La inversión fue ejecutada con offset vertical de 10 puntos bajo el límite superior del modelo ya que la energía debido a las fuentes cerca a la superficie impactan el orden de magnitud en los cálculos de los gradientes y por tanto se afecta la reconstrucción de los parámetros elásticos a medida que se avanza a más profundidad en el modelo, además no se considera aplicar inversión a las zonas CPML (16 puntos de grosor) ya que como fue mencionado estas son un añadido artificial que se usa solo en el procesamiento numérico pero realmente no hacen parte del modelo.

Las pruebas para evaluar el algoritmo implementado involucran dos modelos iniciales, el primero es una imágen con distribución homogénea de los parámetros elásticos ρ, v_p and v_s mientras tanto el segundo fue construido a partir de aplicar un *suavizado* sobre el modelo verdadero utilizando un filtrado gaussiano de 5 puntos de distribución; la idea es observar la evolución en la reconstrucción de los modelos finales desde dos puntos de partida distintos y comprobar la impor-

tancia de una correcta elección del modelo inicial en la FWI para obtener resultados precisos. La figura 24 muestra los modelos iniciales mencionados.

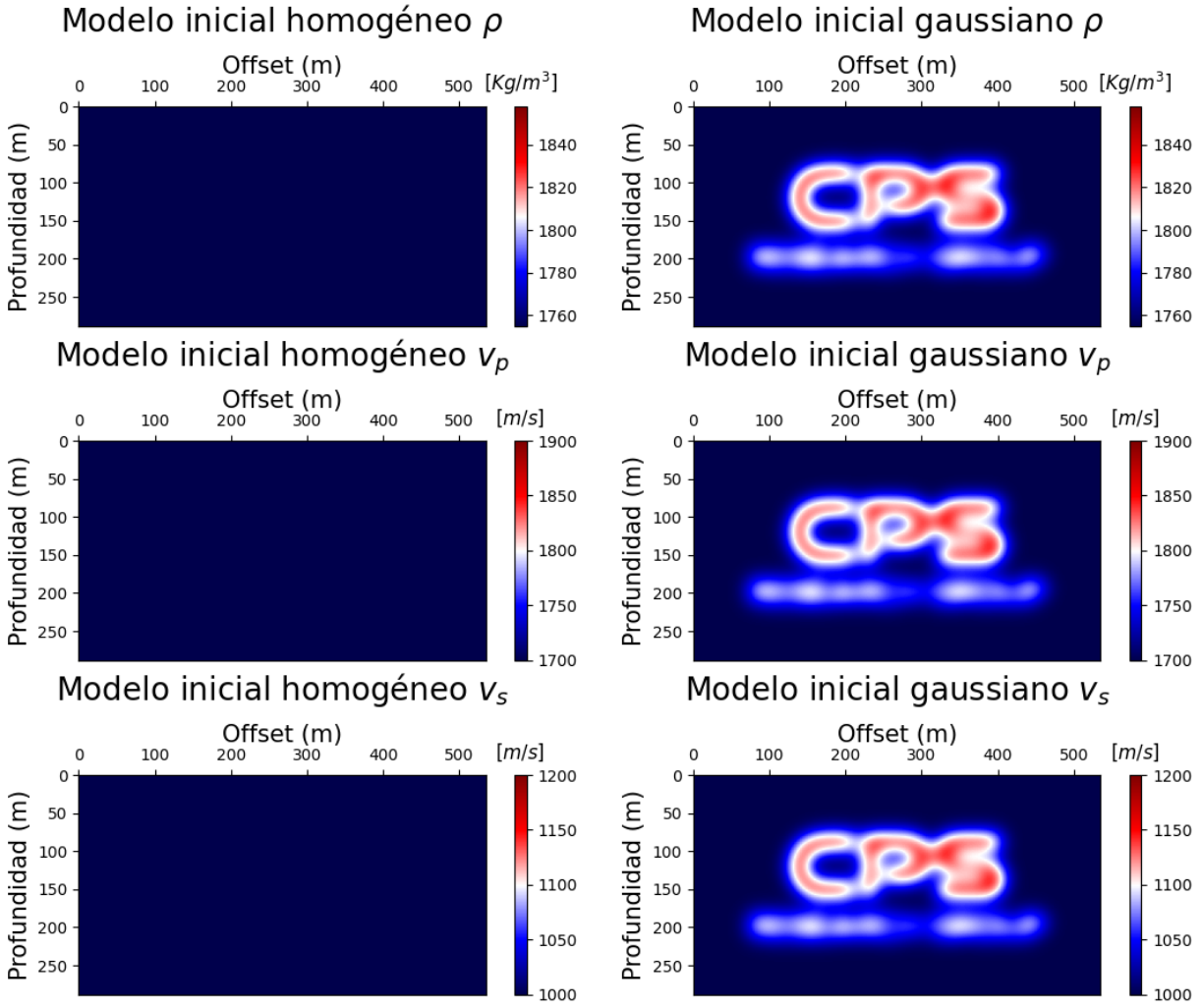

Figura 24. Modelos iniciales CPS para las pruebas de FWI, el modelo inicial homogéneo (izquierda), el modelo inicial filtrado gaussiano (derecha).

Todas las pruebas de FWI buscan evaluar la evolución del modelo actualizado con un creciente número de iteraciones; es importante mencionar que el parámetro α es muy sensible ya que pequeños cambios en su valor pueden llevar a un rápido incremento en la función de costo y gran-

des variaciones en el modelo que se está actualizando; debido a esto, antes de los resultados finales mostrados en esta tesis se tuvo que realizar una serie de pruebas previas variando los valores de α_λ, α_μ y α_ρ con el fin de conseguir el punto en el que efectivamente la función de costo mostrara un comportamiento decreciente monotónico durante toda la inversión.

Otra importante aclaración se debe realizar, existen algunas aplicaciones de FWI elástica que invierten solo uno o dos parámetros manteniendo el tercero en la respuesta esperada en búsqueda de mejorar la calidad de los resultados o evaluar quizá la influencia de cada parámetro de forma independiente, sin embargo esta es una consideración en cierta medida sesgada a la hora de implementar FWI ya que al tener un parámetro en la respuesta la reconstrucción de los otros dos es mas sencilla para el algoritmo, además al trabajar de esta forma no considera la influencia existente entre los 3 parámetros de forma simultánea y es una condición alejada de la realidad pues en una adquisición real difícilmente se puede tener apriori la respuesta de uno de los parámetros elásticos; con base en lo mencionado se debe afirmar que la inversión lograda en este trabajo es multiparamétrica que considera la actualización de los 3 parámetros elásticos de forma simultánea.

En la figura 25 , se muestra los modelos finales obtenidos despues de implementar FWI elástica partiendo del modelo inicial homogéneo; aquí se detalla como mejora la definición de la imágen cuando aumenta las iteraciones, esta reconstrucción es particularmente buena en la definición de las grandes estructuras (las letras CPS) mientras que con las pequeñas (Research group) el resultado presenta menor definición. Además no solo las formas se definieron sino tambien la tendencia hacia los valores esperados de los parámetros que se identifica al observar la intensidad de los colores al comparar este resultado con el modelo verdadero.

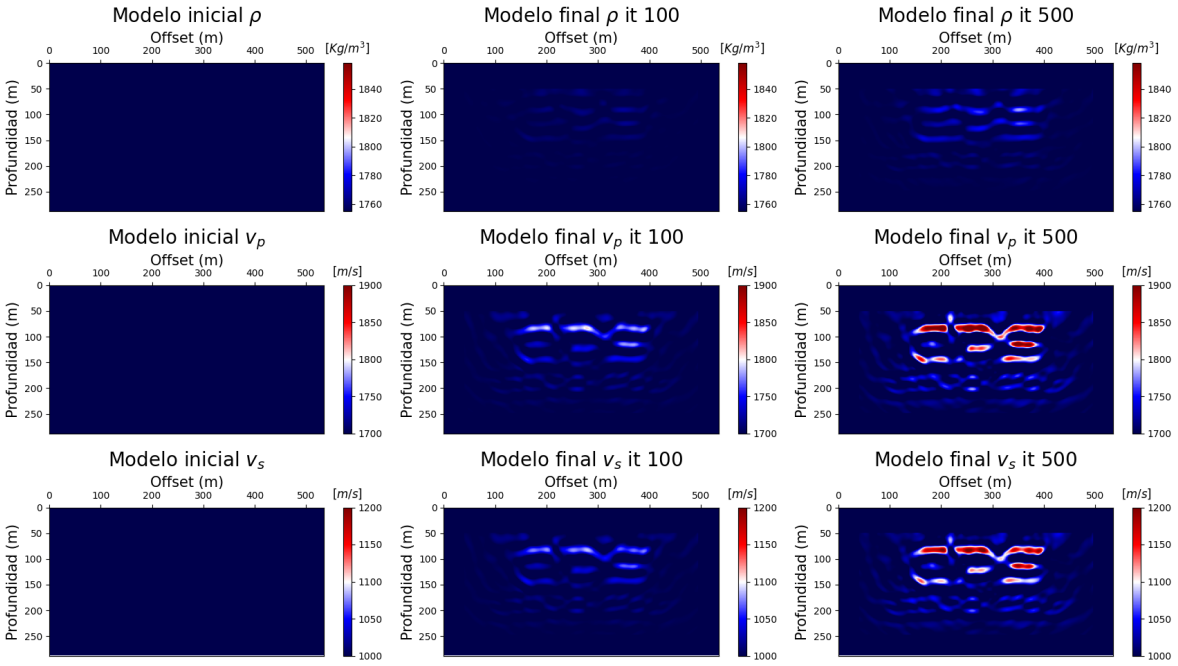

Figura 25. Resultados de la FWI partiendo del modelo inicial homogéneo (izquierda), modelo final después de 100 iteraciones (centro), modelo final después de 500 iteraciones (derecha).

De la imágen se puede apreciar que cada parámetro is actualizado de forma diferente con una mas rápida reconstrucción en v_p y v_s y una mas lenta e incluso menos clara en ρ.

Hasta aquí la evaluación ha sido puramente cualitativa, sin embargo, es posible utilizar métricas cuantitativas como se mencionó al principio del capítulo; la figura 26 (izquierda) muestra el valor MSE calculado para cada parámetro elástico en la reconstrucción de FWI para 500 iteraciones, esta métrica mide que tan alejado esta el modelo actualizado del verdadero y se espera que la tendencia sea hacia la disminución de su valor con el aumento de iteraciones.

A partir de la figura se puede observar que cada parámetro es actualizado a un ritmo diferente especialmente v_s muestra el mejor comportamiento, sin embargo no necesariamente implica que los otros parámetros vayan por mal camino ya que está métrica es calculada como un valor

promedio, que solamente indica que en determinada iteración el modelo final está en *promedio* mas cerca del modelo verdadero pero nada mas puede ser concluido. A pesar de esto, la interpretación de esta métrica es útil para confirmar los resultados obtenidos en la figura 25.

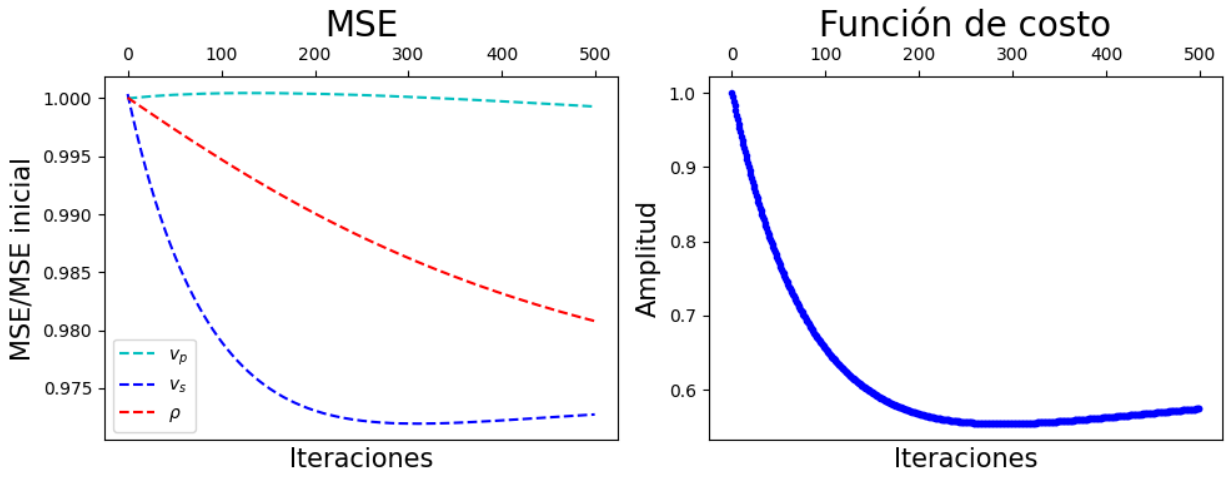

Figura 26. Métricas para evaluar el ajuste de la FWI partiendo del modelo inicial homogéneo, MSE (izquierda), función de costo (derecha).

La otra métrica utilizada es la función de costo y se muestra en la figura 26 (derecha), en este caso se mide la evolución de la diferencia entre el sismograma modelado y el verdadero (residuales) para todos los receptores utilizados; de forma similar a la MSE, en este caso también se espera una tendencia decreciente con el aumento del número de iteraciones.

La tendencia en la función de costo debería disminuir hasta un punto donde se queda aproximadamente estable ya que en este punto se espera que haya alcanzado un mínimo de la función, sin embargo encontrar este mínimo y el comportamiento de línea plana en las últimas iteraciones depende de muchos factores entre ellos que tan lejos el modelo inicial está del verdadero, el valor α seleccionado entre otros; es aquí donde se refleja el problema mal puesto que es la FWI.

A pesar de lo mencionado sobre los problemas relacionados con el modelo inicial, la evolución de la función de costo observada en la figura 26 sugiere que probablemente se ha obtenido la mejor respuesta posible para el modelo inicial seleccionado y continuar con mas iteraciones en el proceso de inversión no llevará a mayores mejoras. Por tanto si se quiere una mejor reconstrucción la mejor opción es elegir un mejor modelo inicial.

La segunda prueba que fue realizada parte de un mejor modelo inicial obtenido con el *suavizado* del modelo verdadero utilizando un filtrado gaussiano de 5 puntos de distribución, así se puede asegurar que este modelo es mejor que el homogéneo. La figura 27 muestra los resultados de la inversión con este modelo inicial.

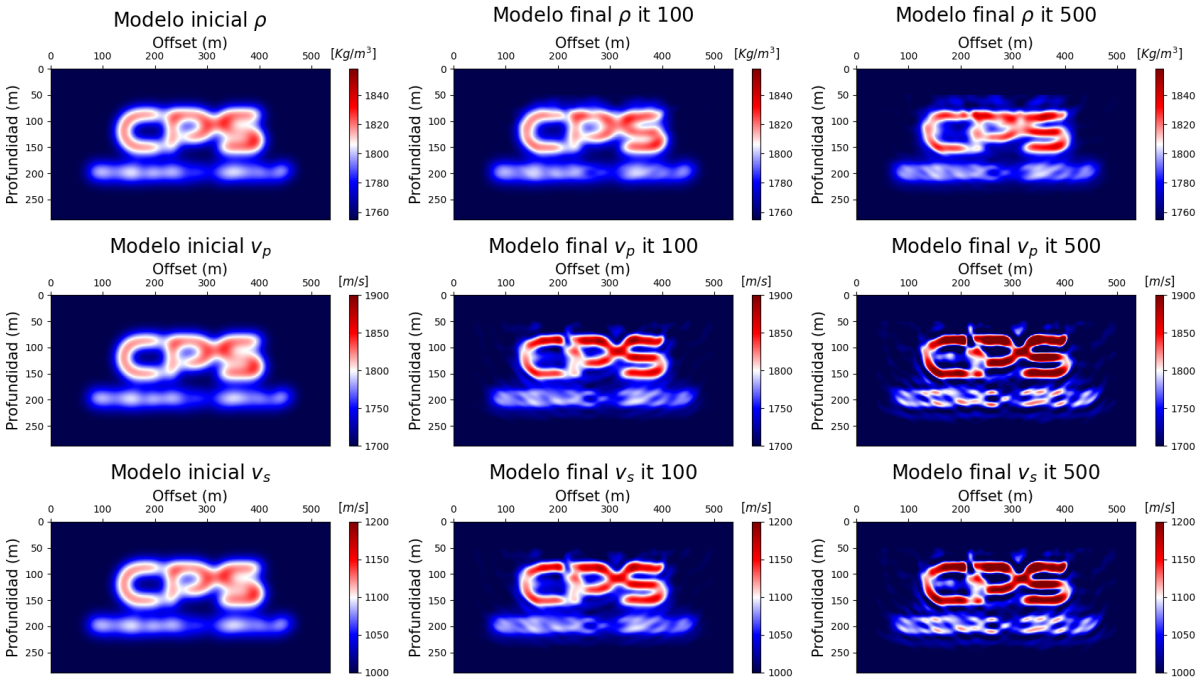

Figura 27. Resultados de la FWI partiendo del modelo inicial filtrado gaussiano (izquierda), modelo final después de 100 iteraciones (centro), modelo final después de 500 iteraciones (derecha).

Las pruebas se ejecutaron para el mismo número de iteraciones que en la figura 25 con el fin de poder comparar los modelos resultantes. Se puede observar como los resultados con modelo inicial filtrado gaussiano mejoran de gran manera al definir de mejor forma las estructuras grandes en el logo CPS y con una mejor intensidad del color que indica el valor del parámetro elástico; las velocidades muestran nuevamente un mejor comportamiento que la densidad.

Una mención especial sobre la reconstruccón de las pequeñas letras *Research Group* donde en este caso logran una mejor defnción especialmente en las letras **CH** en la palabra *Research*, incluso el valor del parámetro casi alcanza el valor verdadero en varias zonas; este comportamiento no se puedo detallar en la FWI que parte del modelo homogéneo. Otra detalle a resaltar en los resultados de la reconstruccón es la mejora en la definición del parámetros ρ tanto en forma como en valor.

La mejora en el modelo final obtenido se confirma al observar el MSE y la función de costo de la figura 28 (izquierda) donde para el caso de MSE se muestra una mejor evolución de los parámetros v_s y ρ comparado con los resultados obtenidos en la figura 26. Igualmente, si se considera la función de costo a la derecha de la imágen la tendencia también ha mejorado de hecho se puede encontrar la zona plana al final de las últimas iteraciones de la inversión que puede indicar que se ha logrado alcanzar un mínimo.

Finalmente, en la figura 29 se presenta una comparación sobre los mejores modelos obtenidos con la FWI partiendo del modelo inial homogéneo y del filtrado gaussiano para un total de 500 iteraciones. Al detallarlos visualmente y considerando el valor de las métricas alcanzado para cada caso es fácil ahora entender la importancia en la elección del modelo inicial para la obtención

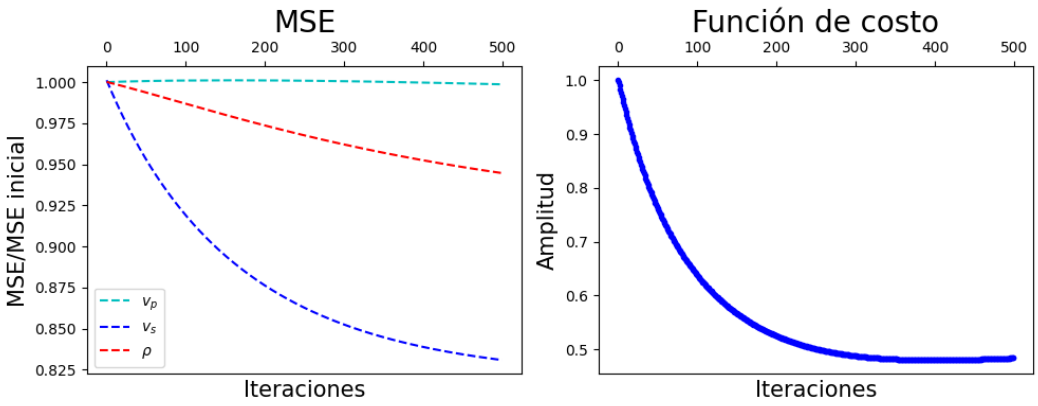

Figura 28. Métricas para evaluar el ajuste de la FWI en el modelo CPS partiendo del modelo inicial filtrado gaussiano, MSE (izquierda), función de costo (derecha).

de modelos con mejor definición y resolución al aplicar FWI elástica.

Una observación importante respecto a los resultados de la FWI es que los parámetros se actualizan a diferente ritmo lo que significa que siempre uno responderá mejor que los otros incluso llegando a estados donde 2 de los parámetros estan cerca de la respuesta y 1 esta bastante alejado, sin embargo este comportamiento no puede ser anticipado.

El modelo considerado en esta primera prueba es bastante sencillo pues contiene solo dos capas, en aplicaciones reales es común encontrar modelos de muchas capas y diferentes valores de parámetros lo que aumenta la dificultad de la FWI para resolver los modelos y genera un aumento en los requerimientos computacionales.

6.1.2. Medición del desempeño computacional de la FWI. Junto con el análisis del nivel de ajuste de los modelos finales, es importante también presentar algunas métricas para evaluar el desempeño computacional de la estrategia programada. Existen varias métricas en la literatura como el tiempo de ejecución, consumo de memoria, *throughput* y otras. La intención

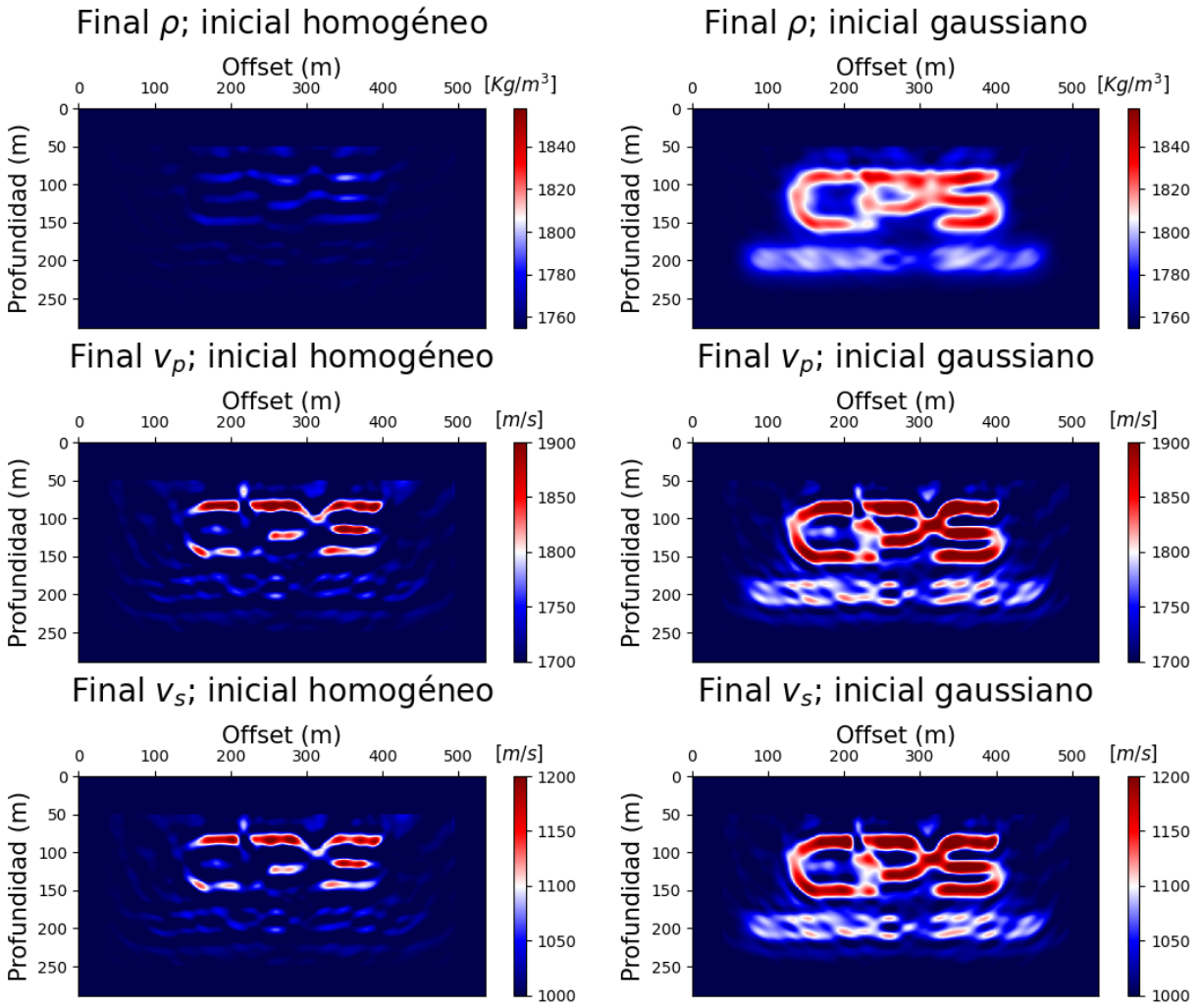

Figura 29. Modelos finales CPS resultado de implementación de FWI, partiendo de modelo inicial homogéneo (izquierda), partiendo de modelo inicial filtrado gaussiano (derecha).

en este trabajo no es presentarlo como una alternativa de optimización si no mas bien como una implementación funcional de referencia de la FWI elástica que explota algunas ventajas de la programación heterogénea en GPU.

Desde esta óptica las métricas mas adecuadas para evaluar el rendimiento de este tipo de algoritmos son el tiempo de ejecución visto como el tiempo total invertido para ejecutar todo

el proceso FWI. La *compute utilization* o *GPU occupancy* en algunos contextos, esta mide el porcentaje de tiempo que cualquier kernel esta ejecutando código y por tanto se mantiene ocupada la GPU, este valor no incluye el tiempo invertido en transferencias de memoria, carga de datos entre otros; al final esta métrica es útil para determinar que cantidad de tiempo realmente se mantuvo trabajando la GPU y no en otras tareas misceláneas.

La última métrica es el consumo de VRAM para la implementación de FWI con esta estrategia, el conocer este valor es importante ya que permite estimar por ejemplo el tamaño de los modelos que pueden ser procesados con este código que es uno de las grandes desventajas cuando se trabaja sobre GPU debido a la limitada cantidad de memoria disponible.

Para realizar las mediciones se utilizará una herramienta conocida como *Nvidia Visual Profiler* ya que es un software desarrollado directamente por el fabricante de las GPU para estos propósitos y permite además presentar la información de forma gráfica facilitando su interpretación. La figura 30 muestra la captura de la interfaz de la herramienta para una medición realizada sobre el algoritmo de FWI para una iteración corriendo sobre el modelo CPS.

La figura presenta en el centro una linea de tiempo donde se localiza el orden de ejecución de los kernel en la implementación FWI, a la izquierda se detallan alrededor de 31 de estos kernels pero se muestran aquellos que tienen mayor impacto. De la figura se puede obtener el valor de la métrica *computing utilization* que es de 93.5 % y el tiempo de ejecución para una iteración FWI en el modelo CPS que lleva a un valor de 13.18 [s] aproximadamente.

Para este valor de tiempo de ejecución por iteración es posible extrapolar por el número de iteraciones para obtener el tiempo total de ejecución del algoritmo corriendo sobre el modelo CPS,

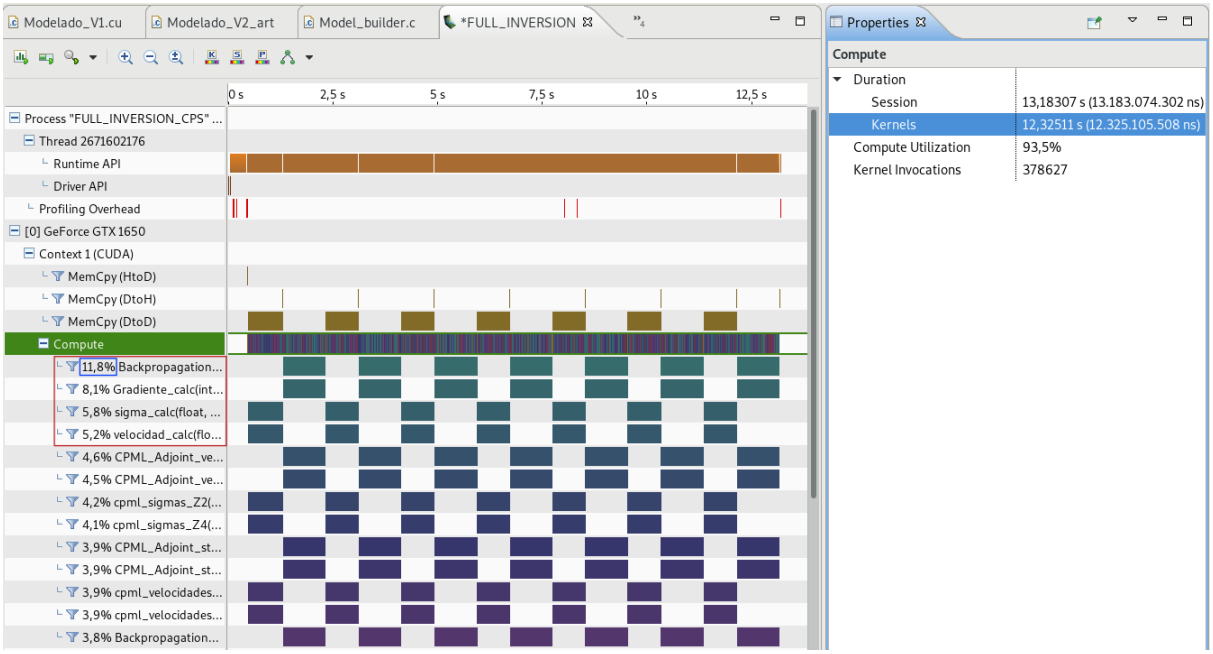

Figura 30. Captura del Nvidia Visual Profiler recolectando información sobre el desempeño de la estrategia FWI implementada sobre el modelo CPS.

al operar de esta forma se obtiene el tiempo de ejecución para la inversión de 100 iteraciones de 23 [min] y para 500 iteraciones toma alrededor de 1.55 [h].

En la figura 30 a la izquierda se resaltó en azul el lugar donde se obtiene el valor de *computing utilization* por cada kernel, de igual forma se resalta en rojo los 4 kernels que consumen mayor cantidad de tiempo en el proceso completo. Esta información es útil para determinar posibles puntos de optimización del código aplicando nuevas técnicas computacionales por ejemplo.

Para obtener el valor de consumo de memoria VRAM se utilizará otra herramienta conocida como *Nvidia-smi* pero con el visualizador gráfico *nvtop*, este programa permite verificar el estado de la GPU y presenta una serie de variables de estado incluyendo el valor de la memoria que cierto algoritmo está utilizando. La figura 31 muestra la interfaz del programa, en esta imágen se resalta

en cuadros rojos las zonas donde se puede leer el consumo de memoria del algoritmo de FWI en GPU.

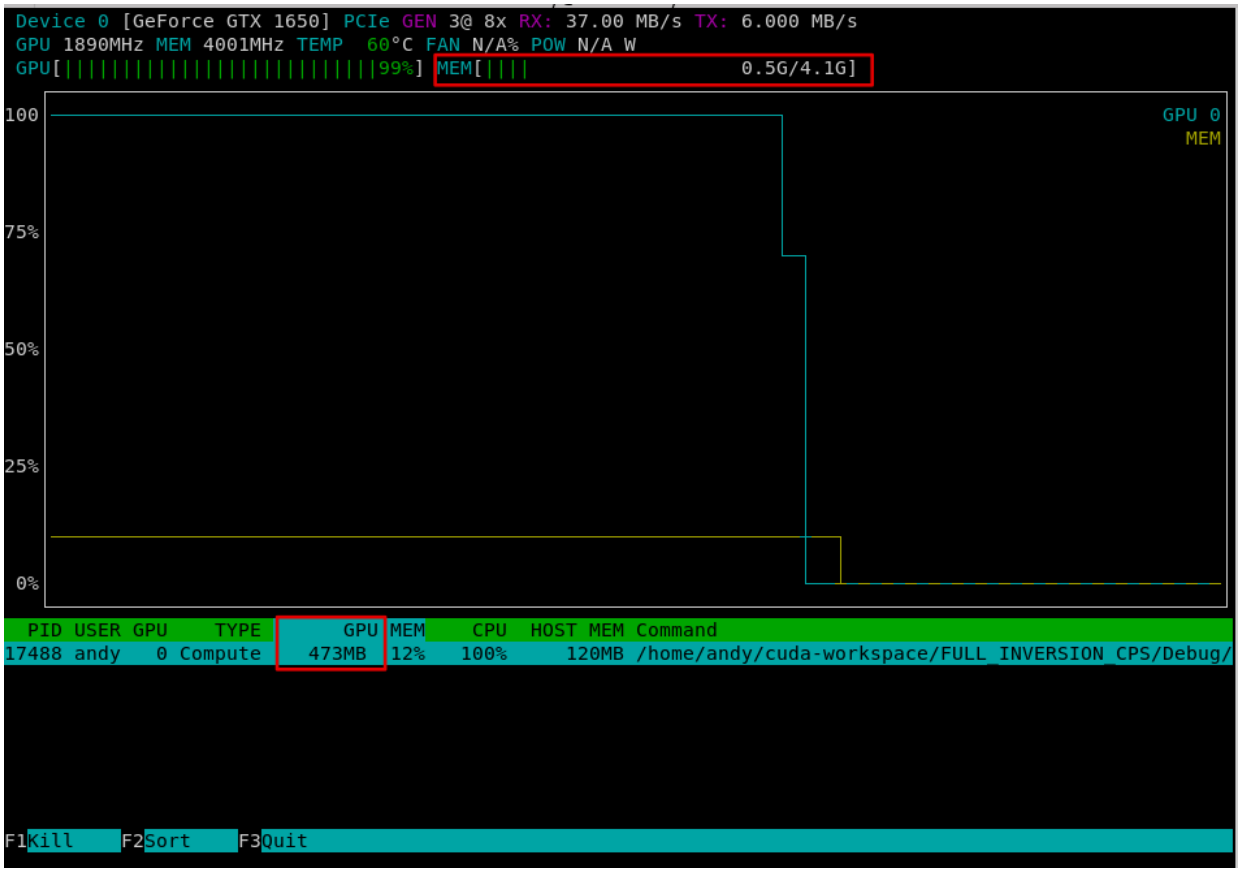

Figura 31. Captura de Nvidia nvtop para determinar la cantidad de memoria VRAM consumida por la implementación FWI ejecutando sobre el modelo CPS.

Para la FWI ejecutada sobre el modelo CPS el consumo total de memoria es de 473 [MB] con un modelo de 214x115 [puntos] y un tiempo de simulación de 1.5 [s] que corresponden a 2000 muestras de tiempo, la mayoria de esta memoria se consume en el almacenamiento de los cubos de propagación para los campos v_x y v_z necesarios para ejecutar la inversión de forma apropiada y en el almacenamiento de los sismogramas.

Para realizar una estimación del valor de memoria que consumen estos elementos dentro del algoritmo se propone dos fórmulas, la primera mostrada en la ecuación 38 permite estimar el consumo de los cubos de propagación utilizados en la inversión, y la segunda fórmula 39 permite estimar el espacio para almacenar los sismogramas verdaderos en GPU.

$$Mem_{cub} = \frac{N_x \times N_z \times 4 \times t_{samples}}{10^6} \times \#wavefields\ saved, \tag{38}$$

$$Mem_{sources} = \frac{\#\ receivers \times 4 \times t_{samples}}{10^6} \times \#\ sources \times 2. \tag{39}$$

Con estas fórmulas, para el caso de la FWI sobre el modelo CPS el valor de la memoria consumida por estos elementos es de 420 [MB] aproximadamente que representa alrededor del 88 % de la memoria consumida por todo el algoritmo, el restante se utiliza en variables intermedias y temporales necesarias para la correcta ejecución.

Es importante mencionar que los valores de esas métricas en especial el tiempo de ejecución y el consumo de memoria no son fijos, ellos varían dinamicamente dependiendo de las dimensiones del modelo, los parámetros de discretización en especial Δt, el número de muestras de tiempo y otros mas; por tanto la selección de estos parámetros en el algoritmo impacta directamente en el desempeño total y deben elegirse cuidadosamente.

Finalmente, las métricas obtenidas aplican de forma similar para la ejecución de FWI partiendo del modelo homogéneo y filtrado gaussiano ya que el algoritmo usado es el mismo, la única diferencia es el modelo inicial que al tener las mismas dimensiones no impacta en las mediciones.

6.2. Modelo Overthrust 2D

6.2.1. Medidas de ajuste para la FWI. Las pruebas con el modelo CPS fueron útiles para determinar que el algoritmo FWI programado es funcional, también sirvió para ver la importancia de la selcción apropiada del modelo inicial para lograr modelos finales de parámetros elásticos mas precisos. Igualmente se realizó la captura de las métricas computacionales que permite estimar las limitaciones en las dimensiones del modelo, parámetros de discretización entre otros.

El modelo CPS se considera un modelo sencillo para saber si el algoritmo funciona, sin embargo, en adquisiciones reales los modelos a reconstruir suelen ser mucho mas complicados, estos pueden incluir muchas capas con diferentes valores para los parámetros en distintas formas geométricas incluyendo estructuras suaves y puntiagudas; todas esas características aumentan la complejidad del modelo y la dificultad del algoritmo para lograr reconstrucciones apropiadas afectando de paso los requerimientos computacionales.

En esta sección se implementará FWI elástica sobre un modelo complejo conocido como *Overthrust 2D* que se encuentra en la literatura usualmente como modelo de pruebas de algoritmos de inversión; este tipo de modelos con estas características se encuentran fácilmente en adquisiciones reales y por tanto usarlo puede dar un idea del desempeño de la estrategia implementada en casos reales.

En la figura 32 se muestra la imágen del modelo verdadero Overthrust, este modelo tiene unas dimensiones de $N_x = 532$ por $N_z = 202$ puntos, el paso espacial de discretización $\Delta h = 12.5$

[m] y paso temporal $\Delta t = 1.2$ [ms] lo que nos lleva a un modelo de 6650 [m] por 2525 [m], con la selección de los parámetros realizada se da cumplimiento a los criterios de estabilidad numérica y dispersión, la inversión se ejecutó con 12 sismogramas verdaderos procedentes de 12 fuentes ubicadas a 37.5 [m] (3 puntos) de profundidad, y distribuidas iniciando en 850 [m] (68 puntos) en la dirección *offset* y cada fuente esta separada 450 [m] (36 puntos); se utiliza la ondícula Ricker de frecuencia central 6 [Hz].

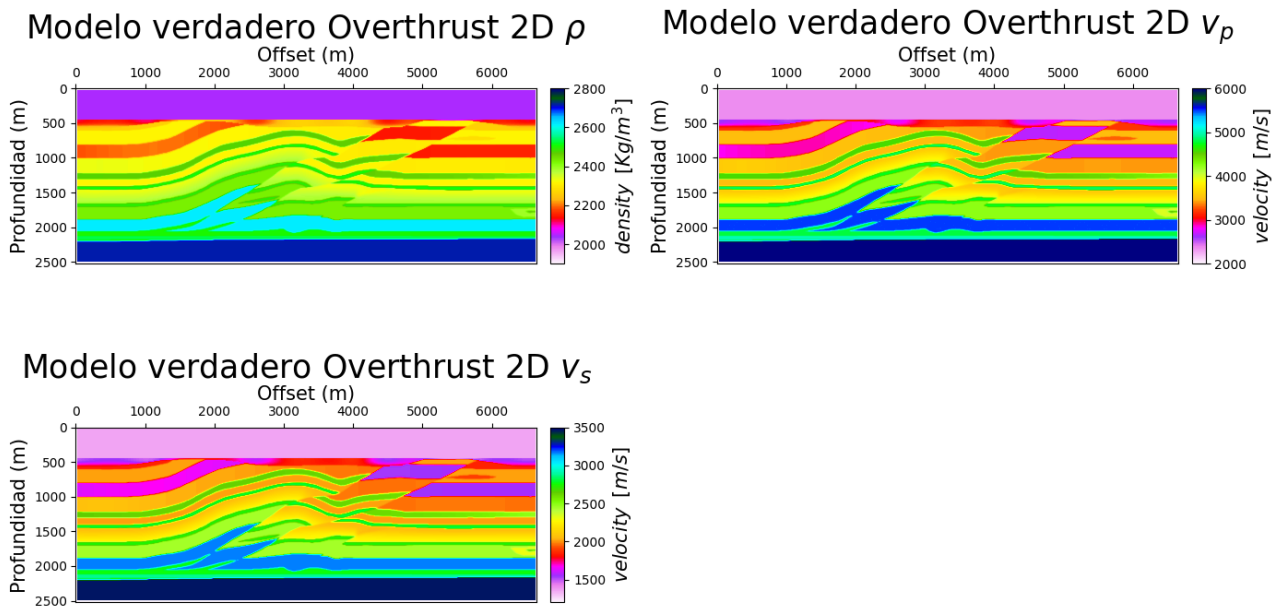

Figura 32. Modelo verdadero sintético Overthrust 2D para pruebas de FWI.

El número de fuentes usadas en este caso se ha incrementado debido a la complejidad del modelo para poder lograr resultados de los modelos finales apropiados en tiempos de simulación realistas, sin embargo, es importante mencionar que la mayoría de procesos de inversión en la literatura suelen considerar cientos de fuentes en el procesamiento lo que impacta grandemente en

los tiempos de ejecución de los mismos pero también permite obtener modelos finales con mejor resolución y calidad; debido a las limitantes en el equipameinto con el que se cuenta se prefiere hacer un balance entre calidad y tiempos de ejecución aceptables lo que lleva a la elección de trabajar con 12 sismogramas.

Los valores de los parámetros elásticos oscilan desde 2047 a 2717 [Kg/m^3] para la densidad ρ, 2360 a 6000 [m/s] para la velocidad v_p y 1360 a 3464 [m/s] para la velocidad v_s; la FWI se ejecutará para un total de 2000 muestras de tiempo y un tiempo de simulación de 2.4 [s]. Los sismogramas modelados se obtienen por un grupo de receptores ubicados a 50 [m] (4 puntos) de profundidad y separados cada 12.5 [m] en la dirección *offset*, los receptores miden las velocidades v_x y vz y ambos sismogramas se consideran en el procesamiento dentro de la FWI.

La inversión se ejecuta con un offset vertical de 30 puntos bajo el límite superior del modelo y no se planea invertir estas zonas ni aquellas asociadas a las CPML (32 puntos de grosor). Para esta prueba se utilizará como modelo inicial solamente el modelo filtrado gaussiano con 5 puntos de dispersión debido a que por la complejidad del modelo Overthrust 2D se requiere un mejor punto inicial.

La figura 33 presenta a la izquierda el modelo inicial filtrado gaussiano y a la derecha el modelo final después de aplicar FWI sobre el modelo inicial para un total de 800 iteraciones.

Se puede observar de los resultados la mejora en la definición de las estructuras especialmente en los parámetros v_p y v_s comparado con el modelo inicial, sin embargo nuevamente el parámetro ρ se queda un poco rezagado aunque se detecta también mejoría. Se debe prestar especial atención a las estructuras puntiagudas alrededor entre los 4000 y 5000[m] en dirección *offset*

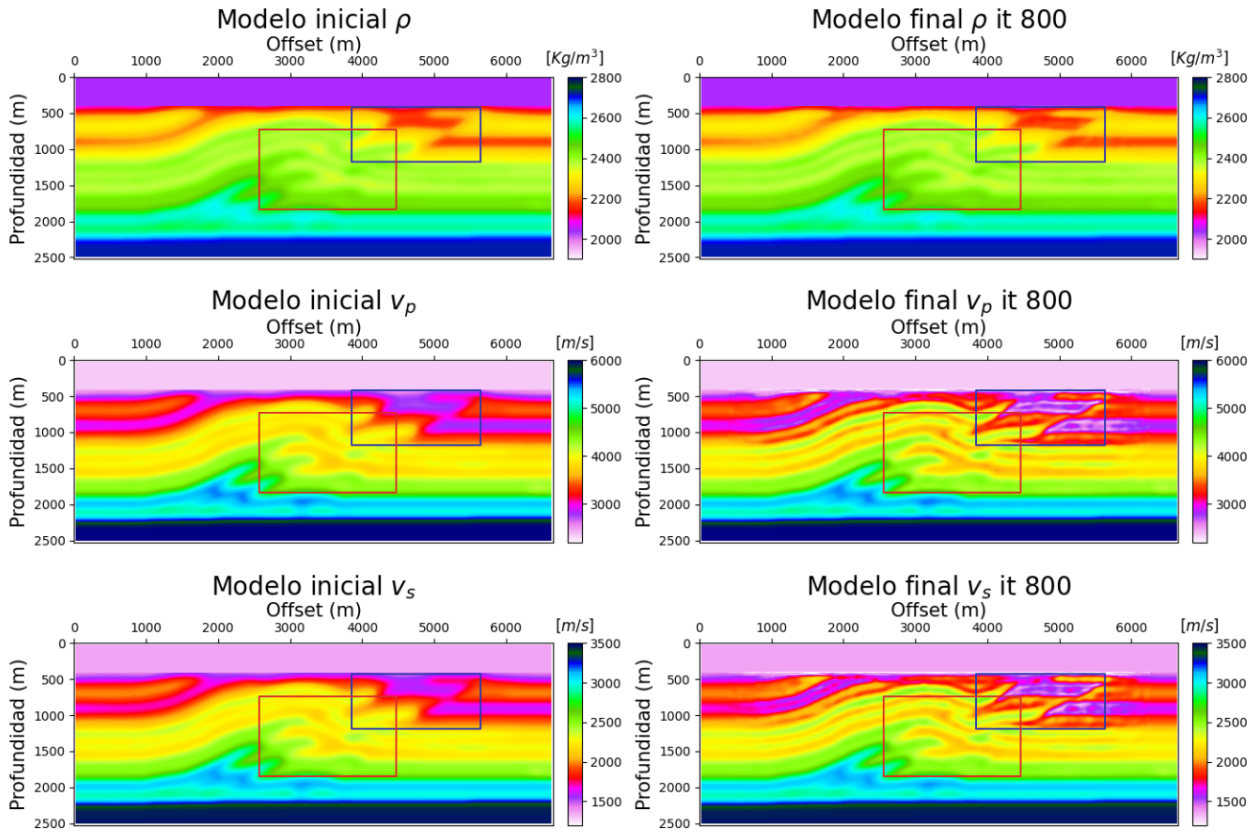

Figura 33. Comparación de la inversión para el modelo Overthrust 2D: modelo inicial filtrado gaussiano (izquierda), modelo final después de 800 iteraciones (derecha).

y el nivel de ajuste de los colores aquí y sobre los 1000 y 2000 [m] a una profundidad de entre 500 a 1500 [m].

Se puede apreciar igualmente una tendencia de mejora en la definición de las estructuras y los valores inicialmente sobre las zonas superiores y cercanas a la superficie del modelo avanzando de forma mas lenta en la medida que se profundiza, este comportamiento también se pudo evidenciar en el modelo CPS donde la mejor resolución de las estructuras sucedió sobre las letras grandes *CPS* y posteriormente continuaba con las letras mas pequeñas ubicadas a mayor profundi-

dad *Research Group*.

Para poder distinguir las mejoras en la reconstrucción de los modelos, las figuras 34 y 35 presentan una comparación entre el modelo inicial, el modelo final y el modelo verdadero sobre 2 zonas de acercamiento que hacen parte del modelo, una zona superficial y la otra mas profunda, se puede apreciar que existe una mejora tanto en la definición de las estructuras como en el valor del parámetro elástico correspondiente; la zona 1 se resalta con color azul y la zona 2 se resalta con color rojo en la figura 33.

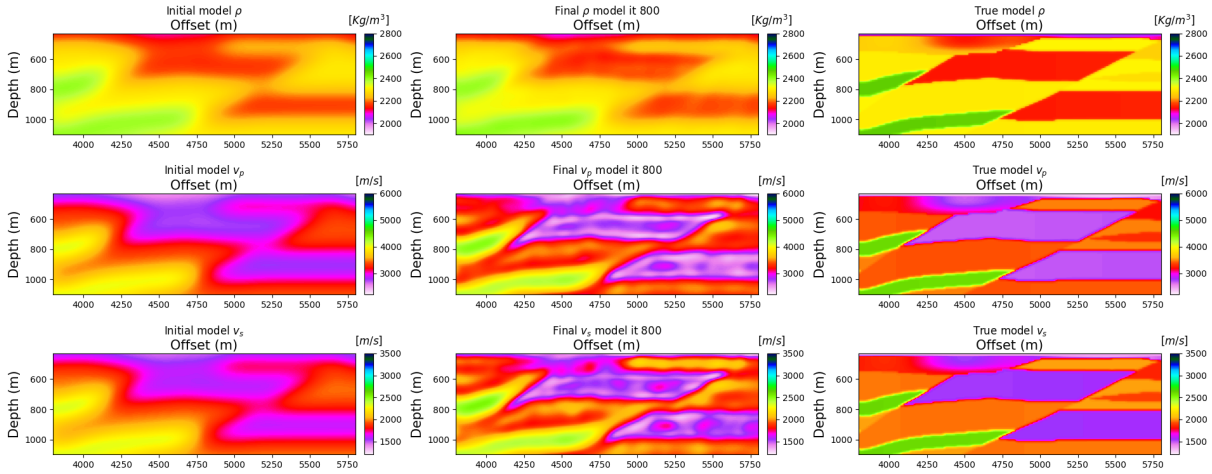

Figura 34. Resultado de la inversión en la zona de acercamiento 1 para la inversión del modelo Overthrust: modelo inicial filtrado gaussiano (izquierda), modelo final después de 800 iteraciones (centro), modelo verdadero (derecha).

De forma similar a la reconstrucción del modelo CPS, en al figura 36 se presenta el MSE y la correspondiente función de costo para los modelos de parámetros elásticos resultado de la inversión. Al observar el MSE se confirma lo que visualmente ya se había detallado y es que

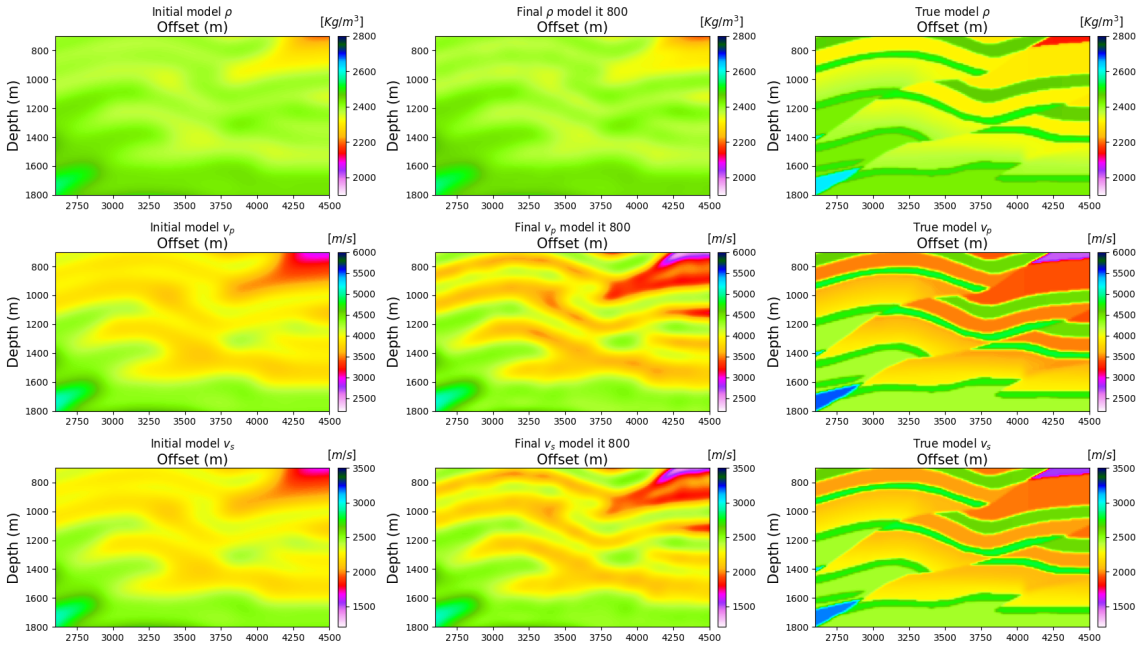

Figura 35. Resultado de la inversión en la zona de acercamiento 2 para la inversión del modelo Overthrust: modelo inicial filtrado gaussian (izquierda), modelo final después de 800 iteraciones (centro), modelo verdadero (derecha).

los mejores resultados se obtiene para los parámetros v_p y v_s sin embargo, el parámetro ρ esta avanzando solo que de forma mas lenta; en general el comportamiento visto en el MSE es el esperado.

Al considerar la función de costo se puede resaltar que un mínimo ha sido alcanzado y por tanto el modelo final es aceptable, a pesar de ello parece que se puede mejorar un poco más con algunas iteraciones adicionales; la pregunta es si vale la pena invertir mas tiempo de ejecución para lograr una mejora que puede no ser apreciable en el modelo final y si puede incrementar de gran manera el tiempo total necesario.

Finalmente en la figura 37 se muestra el modelo final resultado de la implementación de FWI después de 800 iteraciones y el modelo verdadero.

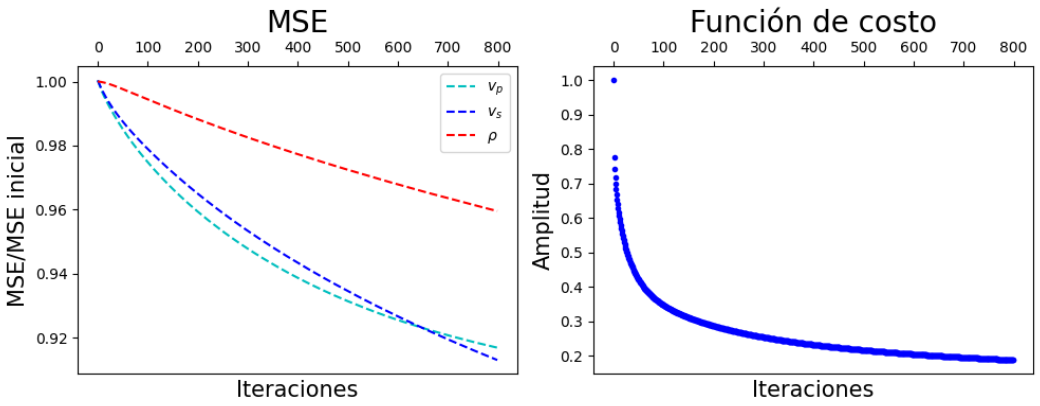

Figura 36. Métricas para evaluar el ajuste de la FWI en el modelo Overthrust 2D partiendo del modelo inicial filtrado gaussiano, MSE (izquierda), función de costo (derecha).

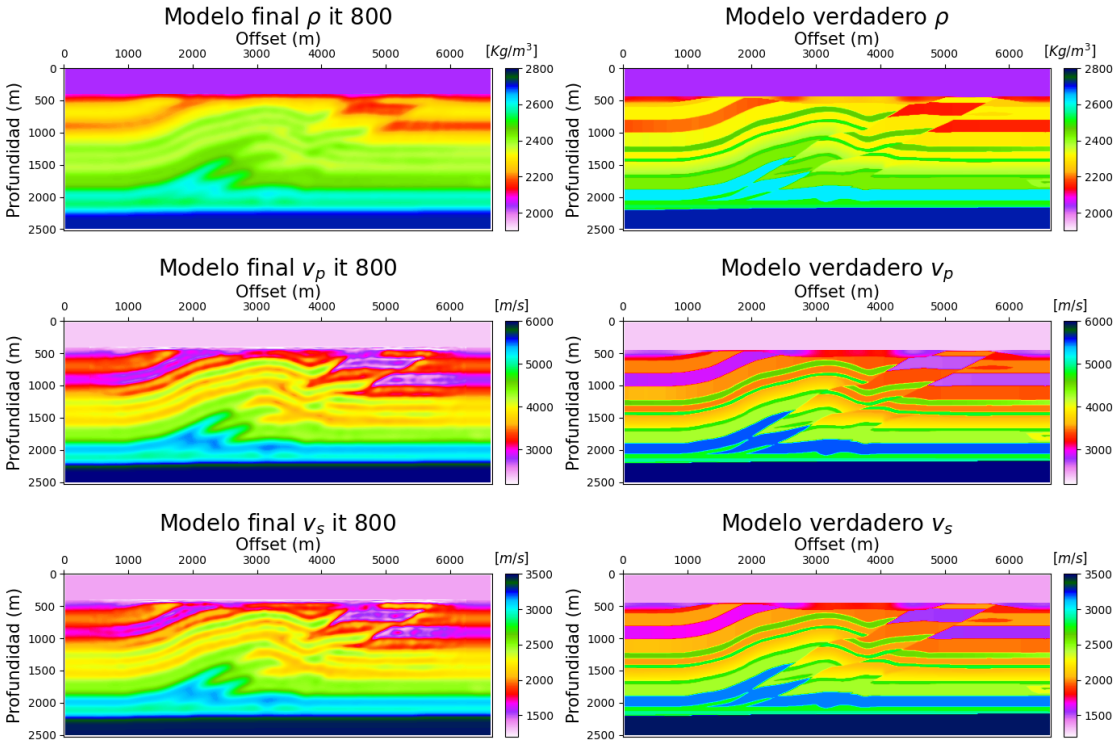

Figura 37. Comparación entre el modelo final después de la FWI de 800 iteraciones (izquierda) y el modelo verdadero (derecha).

6.2.2. Medición del desempeño computacional de la FWI. Para el análisis del desempeño computacional serán utilizadas las mismas métricas presentadas cuando se evaluó el modelo CPS, la figura 38 muestra la captura del Nvidia Visual Profiler que recopila información de *Compute utilization* y tiempo de ejecución, el primer dato llegando a un valor de 97.1 % y el último llegando a un valor de 47.62 [s] por iteración FWI; con estos datos se puede estimar el tiempo total de ejecución para las 800 iteraciones llegando a aproximadamente 11 [h].

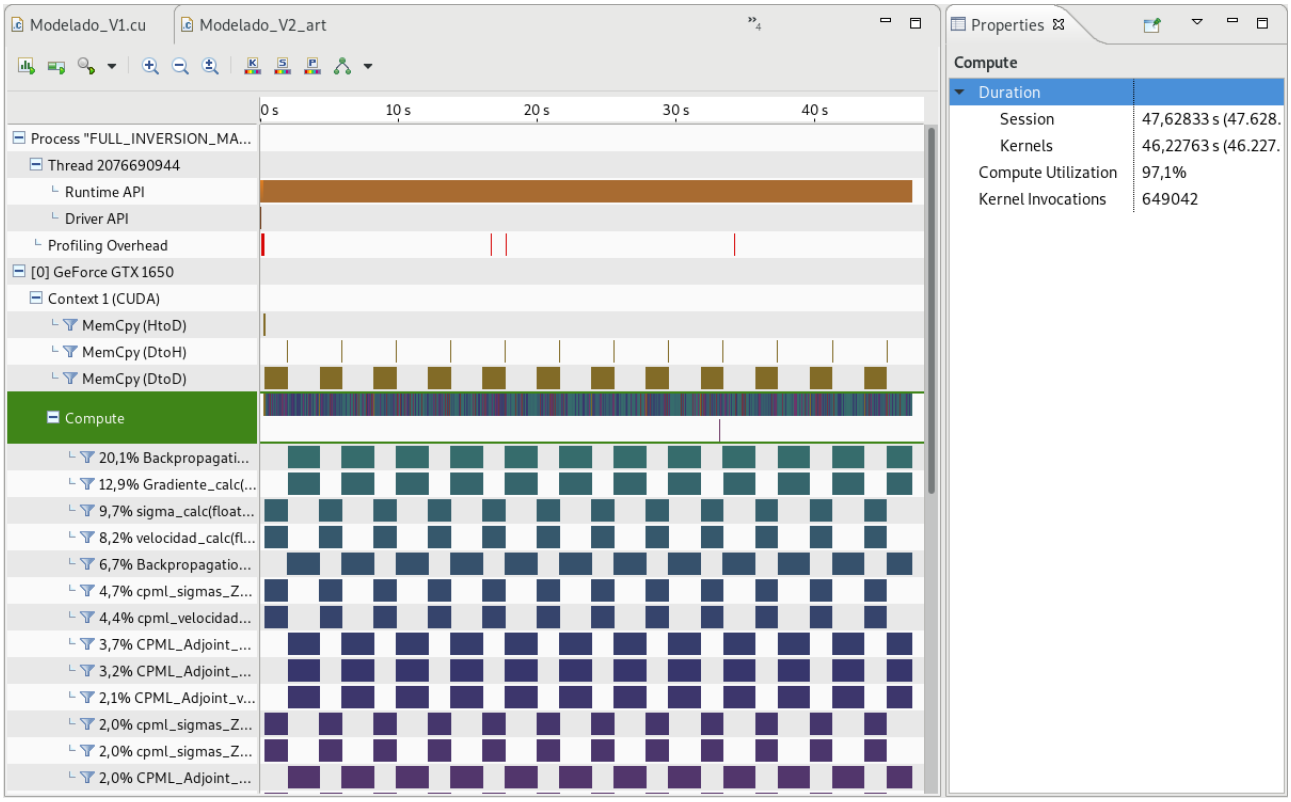

Figura 38. Captura del Nvidia Visual Profiler recolectando información sobre el desempeño de la estrategia FWI implementada sobre el modelo Overthrust 2D.

El consumo de memoria VRAM se obtiene a través de la herramienta nvtop como se muestra en la figura 39, el total de memoria consumida al ejecutar la FWI es de 1901 [MB] para un

modelo de dimensiones 532x202 [puntos] y un tiempo de simulación de 2.4 [s] que corresponden a 2000 muestras de tiempo, nuevamente la mayoría del consumo de esta memoria proviene del almacenamiento de los cubos para los campos v_x y v_z y de los 12 sismogramas verdaderos.

De forma similar a las mediciones realizadas con el modelo CPS, se puede hacer una estimación de los elementos que mas consumen memoria dentro del algoritmo siguiendo las expresiones 38 y 39 dando como resultado el valor teórico de 1822 [MB] que es aproximadamente el 96 % de la memoria total utilizada por el algoritmo dejando el restante para las variables temporales e intermedias.

Al analizar este valor de consumo de memoria y tomando en cuenta que en esta inversión se esta usando la mitad de la capacidad total de memoria de la GPU se hace mas claro la razón por la cual implementaciones de FWI sobre modelos reales necesitan grandes cantidades de recursos computacionales, debido a esto, es usual encontrar implementaciones que explotan el poder no solo de una GPU sino de múltiples configuraciones CPU-GPU en los conocidos clusteres de computo o supercomputadores.

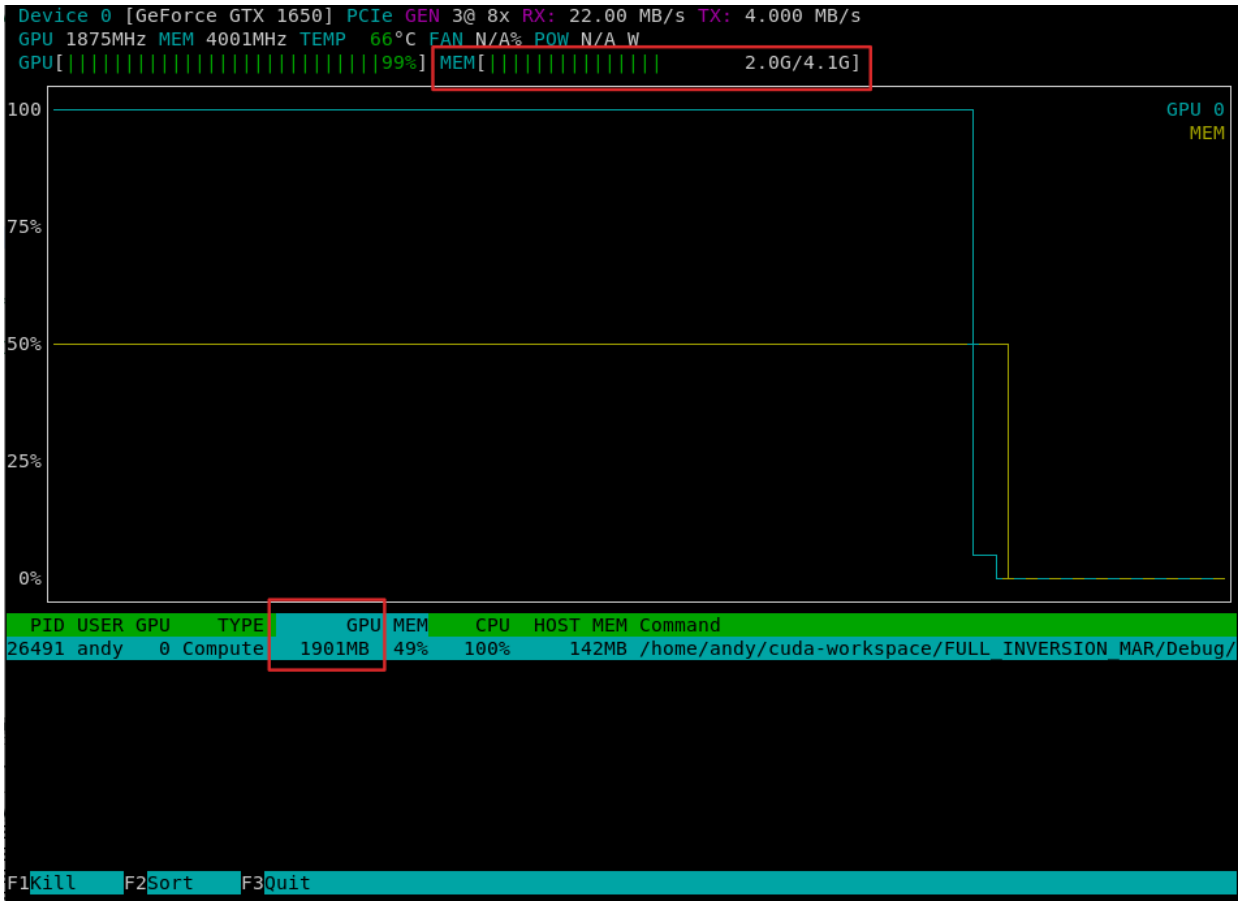

Figura 39. Captura de Nvidia nvtop para determinar la cantidad de memoria VRAM consumida por la implementación FWI ejecutando sobre el modelo Overthust 2D.

7. Conclusiones

La inversión de onda completa (FWI) elástica 2D es un procedimiento de avanzada utilizado para obtener imágenes de alta resolución de la tierra. Sin embargo el procedimiento representa un reto computacional en su implementación debido a la cantidad de información y la complejidad del método. En este trabajo, se desarrolló una estrategia computacional para programar el algoritmo de FWI que aprovecha la nueva computación HPC uniendo fuerzas de paralelismo en GPU y programación secuencial en CPU.

La estrategia propuesta se enfoca principalmente en intentar reducir el consumo de memoria de la FWI y disminuir el impacto en el tiempo de ejecución; para ello se aplicaron 3 tácticas computacionales en el algoritmo: diseño de hilos 2D, Cálculo de CPML con memoria reducida y almacenamiento del cubo de propagación en VRAM.

Con el objeto de evaluar el desempeño de la implementación FWI programada, se consideraron dos modelos diferentes (CPS and Overthrust 2D) con distintos niveles de complejidad y se utilizaron métricas para evaluar el nivel de ajuste y el desempeño computacional, pero también se realizó un análisis visual de los modelos finales obtenidos con la implementación; los resultados muestran que el algoritmo es completamente funcional y que se puede considerar probarlo utilizando modelos reales.

Los resultados obtenidos muestran que no todos los parámetros elásticos son reconstruidos al mismo ritmo, en todas las pruebas siempre hay uno de estos que se retrasa. este comportamiento puede deberse al fenómeno conocido como *cross-talk* que dice que la evolución en la actualización

de un parámetro se ve influenciada de alguna forma por los otros dos, este problema se encuentra bajo constante investigación en la actualidad.

La inversión de onda completa se considera un problema mal puesto que puede llevar a modelos finales inadecuados, al analizar los resultados obtenidos con el modelo CPS se puede resaltar la importancia de la elección de un modelo inicial adecuado lo mas cercano posible al modelo verdadero si se espera obtener resultados precisos y correctos de la implementación de la FWI; sin embargo encontrar modelos iniciales adecuados es un problema en si mismo que necesita mayor investigación.

Se pudo apreciar un comportamiento particular en las pruebas realizadas relacionado con la mejor y mas rápida reconstrucción de los parámetros elásticos en las zonas superficiales del modelo comparado con las zonas profundas, esto puede indicar que el método es mas adecuado para reconstruir modelos cercanos a la superfice sin embargo es necesario mayor investigación al respecto.

El análisis de las métricas de desempeño computacional confirmaron que la FWI es un procedimiento de alto consumo de recursos ya que en todas las pruebas la GPU estuvo trabajando a plena capacidad la mayoría del tiempo, al considerar los resultados especialmente en el modelo Overthrust 2D se vuelve claro que se necesita establecer un balance entre los recursos computacionales, las características del modelo y los parámetros de discretización si se espera obtener modelos finales precisos usando determinado hardware.

Con base en lo mencionado surge otra preocupación asociada al tiempo total y la memoria invertida para completar la ejecución de la FWI, con un modelo como el Overthrust 2D y un

número reducido de sismogramas (solo 12) el procedimiento tomo 11 horas y ocupó alrededor de 2GB de memoria VRAM para lograr un resultado adecuado. Ahora, si se tiene en cuenta que adquisiciones de campo normalmente contienen cientos de fuentes para mejorar la resolución de los modelos finales, esto lleva a pensar que quizá el tiempo invertido o el hardware disponible pueda no ser ni adecuado ni suficiente en algunos casos y se deban buscar mejores implementaciones sobre por ejemplo clusteres de cómputo.

Todos estas inquietudes abren la puerta para buscar nuevas estrategias de implementación en futuros trabajos que exploten por ejemplo nuevas técnicas de optimización con referencia al trabajo ya realizado y presentado en esta tesis a fin de mejorar el desempeño del método y alcanzar resultados apropiados en tiempos realistas y con menor consumo de recursos computacionales.

7.1. Trabajo futuro

Ya que la estrategia de implementación de FWI en este trabajo utilizó el método de gradiente descendente como procedimiento de actualización de los parámetros elásticos, podría ser útil mejorar el algoritmo utilizando otro procedimiento con mejor radio de convergencia como el gradiente conjugado, Newton o métodos quasi-Newton; se hace mención especial a LBFGS que ayuda a alcanzar modelos finales usando menor cantidad de iteraciones y por tanto menor tiempo de ejecución.

Para mejorar los resultados finales en la implementación de la FWI, el algoritmo se puede mejorar realizando un aproximación multiescala de la FWI y utilizar mayor cantidad de sismogramas para contar con mayor información a ser procesada.

Se puede aplicar técnicas de optimización a nivel de la arquitectura de GPU en el algoritmo

para mejorar el desempeño de la estrategia FWI, estas técnicas pueden incluir por el ejemplo el uso de memoria unificada, la utilización de los *tensor cores* especializados cálculos matriciales, explorar mas profundamente la configuración de memorias de la GPU y utilizar los registros o la memoria compartida para asignar recursos de forma manual solo por mencionar algunas opciones.

Evaluar el algoritmo de FWI elástica 2D presentado usando datos de una adquisición real.

Referencias Bibliográficas

Abreo, D. L., Abreo, S. A., and Ramirez, A. B. (2017). A hybrid methodology for a 3d full waveform inversion in time domain using gpus. In *15th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 31 July-3 August 2017*, pages 369–373. Brazilian Geophysical Society.

Arora, M. (2012). The architecture and evolution of cpu-gpu systems for general purpose computing.

Brittan, J., Bai, J., Delome, H., Wang, C., and Yingst, D. (2013). Full waveform inversion â€“ the state of the art. *First Break*, 31(10):75–81.

Carcione, J. M., Herman, G. C., and ten Kroode, A. P. E. (2002). Seismic modeling. *GEOPHYSICS*, 67(4):1304–1325.

Chanaga, A. P., Galvis, I. S., and Silva, A. R. (2020a). Implementation of finite differences 2D elastic wave modeling on a GPU. In Cruz, A., editor, *Avances de las ciencias bÃ¡sicas e ingenierÃa en tiempos de la COVID-19*.

Chanaga, A. P., Silva, A. B. R., and Galvis, I. J. S. (2020b). Computational strategies for implementation of 2D elastic wave modeling in GPU. *Entre Ciencia e IngenierÃa*, 14(28):52–58.

Chen, K. and Sacchi, M. D. (2018). Time-domain elastic Gauss-Newton full-waveform inver-

sion: A matrix-free approach. SEG Technical Program Expanded Abstracts, pages 1208–1212. Society of Exploration Geophysicists.

Cheng, J., Grossman, M., and McKercher, T. (2014). *Professional CUDA C Programming*. John Wiley & Sons.

Crase, E., Pica, A., Noble, M., Mcdonald, J., and Tarantola, A. (1990). Robust elastic nonlinear waveform inversion; application to real data. *Geophysics*, 55(5):527–538.

Crase, E., Wideman, C., Noble, M., and Tarantola, A. (1992). Nonlinear elastic waveform inversion of land seismic reflection data. *Journal of Geophysical Research: Solid Earth*, 97(B4):4685–4703.

Duarte, L., Donno, D., Lopes, R., and Romano, J. (2014). Seismic Signal Processing: Some Recent Advances. pages 2362–2366.

Gao, Y., Song, H., Zhang, J., and Yao, Z. (2017). Comparison of artificial absorbing boundaries for acoustic wave equation modelling. *Exploration Geophysics*, 48(1):76–93.

Gauthier, O., Virieux, J., and Tarantola, A. (1986). Two-dimensional nonlinear inversion of seismic waveforms; numerical results. *Geophysics*, 51(7):1387–1403.

Ginter, S., Liebler, M., Steiger, E., Dreyer, T., and Riedlinger, R. E. (2002). Full-wave modeling of therapeutic ultrasound: Nonlinear ultrasound propagation in ideal fluids. *The Journal of the Acoustical Society of America*, 111(5):2049–2059.

Kadlubiak, K., Jaros, J., and Treeby, B. E. (2018). Gpu-accelerated simulation of elastic wave propagation. In *2018 International Conference on High Performance Computing Simulation (HPCS)*, pages 188–195.

Komatitsch, D. and Martin, R. (2007). An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation. *Geophysics*, 72(5).

Lines, L. R., Slawinski, R., and Bording, R. P. (1999). A recipe for stability of finite-difference wave-equation computations. *Geophysics*, 64(3):967–969.

Menke, W. (2018). *Geophysical Data Analysis*. Elsevier Inc, 4th edition edition.

Moczo, P., Robertsson, J. O., and Eisner, L. (2007). The finite-difference time-domain method for modeling of seismic wave propagation. *Advances in geophysics*, 48:421–516.

Moser, F., Jacobs, L., and Qu, J. (1999). Modeling elastic wave propagation in waveguides with the finite element method. *Ndt & E International - NDT E INT*, 32.

Nickolls, J. and Dally, W. J. (2010). The gpu computing era. *international symposium on micro-architecture*, 30(2):56–69.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

Noriega, R. F., Ramirez, A. B., Abreo, S. A., and Arce, G. R. (2017). Implementation strategies of the seismic full waveform inversion. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 1567–1571. IEEE.

NVIDIA (2018). *Nvidia turing GPU architecture*.

NVIDIA (2020a). *CUDA C++ Best Practices Guide*. Nvidia.

NVIDIA (2020b). *CUDA C++ Programming Guide*. Nvidia.

Plessix., R. (2006). A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysics*, 167:495–503.

Sanchez, I., Duarte, C., and Agudelo, W. (2017). Simulation of surface seismic waves propagation by 2d finite-difference elastic wave modeling in the presence of complex surface topography. In *XI CONGRESO COLOMBIANO DE METODOS NUMERICOS*.

Segovia Spadini, A., Diogo, L., Slob, E., and Prado, R. (2016). Strategies for the application of a conjugate gradient FWI algorithm of shallow environments. pages 1–5.

Seriani, G. and Oliveira, S. P. (2020). Numerical modeling of mechanical wave propagation. *La Rivista del Nuovo Cimento*, 43(9):459–514.

Tarantola, A. (1984). Inversion of seismic reflection data in the acoustic approximation. *Geophysics*, 49(8):1259–1266.

Tarantola, A. (1986). A strategy for nonlinear elastic inversion of seismic reflection data. *Geophysics*, 51(10):1893–1903.

Tarantola, A. (2005). *Inverse Problem Theory and Methods for Model Parameter Estimation*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics.

Virieux, J. (1986). P sv wave propagation in heterogeneous media velocity stress finite difference method. *Geophysics*, 51(4):889–901.

Virieux, J., Asnaashari, A., Brossier, R., Métivier, L., Ribodetti, A., and Zhou, W. (2017). An introduction to full waveform inversion. In *Encyclopedia of exploration geophysics*, pages R1–1. Society of Exploration Geophysicists.

Virieux, J., Etienne, V., Cruz-Atienza, V., Brossier, R., Chaljub, E., Coutant, O., Garambois, S., Mercerat, E., Prieux, V., Operto, S., Ribodetti, A., and Tago, J. (2012). Modelling Seismic Wave Propagation for Geophysical Imaging.

Virieux, J. and Operto, S. (2009). An overview of full-waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC1–WCC26.

Virieux, J., Operto, S., Ben-Hadj-Ali, H., Brossier, R., Etienne, V., Sourbier, F., Giraud, L., and Haidar, A. (2009). Seismic wave modeling for seismic imaging. *The Leading Edge*, 28(5):538–544.

Wang, Y. (2015). Generalized seismic wavelets. *Geophysical Journal International*, 203:1172–1178.

Wesdorp, J, J. (2018). A general recipe for obtaining adjoint equations and gradients and its application to full waveform inversion of 2D isotropic elastic media in finite difference time-domain. *Instituto Colombiano de Petroleo (ICP)-TU-Delft-Netherlands, Internship Report*.

Apéndices

Apéndice A. Discretización por diferencias finitas (FD) de segundo orden en tiempo y cuarto en espacio de las ecuaciones de onda elásticas

$$v_x{}^{n}_{(i+\frac{1}{2},k)} = v_x{}^{n-1}_{(i+\frac{1}{2},k)} + \frac{\Delta t}{\rho_{(i+\frac{1}{2},k)}\Delta h}\left[\frac{9}{8}\left(\sigma_{xx}{}^{n-\frac{1}{2}}_{(i+1,k)} - \sigma_{xx}{}^{n-\frac{1}{2}}_{(i,k)}\right) - \frac{1}{24}\left(\sigma_{xx}{}^{n-\frac{1}{2}}_{(i+2,k)} - \sigma_{xx}{}^{n-\frac{1}{2}}_{(i-1,k)}\right) + \frac{9}{8}\left(\sigma_{xz}{}^{n-\frac{1}{2}}_{(i+\frac{1}{2},k+\frac{1}{2})} - \sigma_{xz}{}^{n-\frac{1}{2}}_{(i+\frac{1}{2},k-\frac{1}{2})}\right) - \frac{1}{24}\left(\sigma_{xz}{}^{n-\frac{1}{2}}_{(i+\frac{1}{2},k+\frac{3}{2})} - \sigma_{xz}{}^{n-\frac{1}{2}}_{(i+\frac{1}{2},k-\frac{3}{2})}\right)\right]$$

$$v_z{}^{n}_{(i,k+\frac{1}{2})} = v_z{}^{n-1}_{(i,k+\frac{1}{2})} + \frac{\Delta t}{\rho_{(i,k+\frac{1}{2})}\Delta h}\left[\frac{9}{8}\left(\sigma_{xz}{}^{n-\frac{1}{2}}_{(i+\frac{1}{2},k+\frac{1}{2})} - \sigma_{xz}{}^{n-\frac{1}{2}}_{(i-\frac{1}{2},k+\frac{1}{2})}\right) - \frac{1}{24}\left(\sigma_{xz}{}^{n-\frac{1}{2}}_{(i+\frac{3}{2},k+\frac{1}{2})} - \sigma_{xz}{}^{n-\frac{1}{2}}_{(i-\frac{3}{2},k+\frac{1}{2})}\right) + \frac{9}{8}\left(\sigma_{zz}{}^{n-\frac{1}{2}}_{(i,k+1)} - \sigma_{zz}{}^{n-\frac{1}{2}}_{(i,k)}\right) - \frac{1}{24}\left(\sigma_{zz}{}^{n-\frac{1}{2}}_{(i,k+2)} - \sigma_{zz}{}^{n-\frac{1}{2}}_{(i,k-1)}\right)\right]$$

$$\sigma_{xx}{}^{n+\frac{1}{2}}_{(i,k)} = \sigma_{xx}{}^{n-\frac{1}{2}}_{(i,k)} + \frac{\Delta t\left(\lambda_{(i,k)}+2\mu_{(i,k)}\right)}{\Delta h}\left[\frac{9}{8}\left(v_x{}^{n}_{(i+\frac{1}{2},k)} - v_x{}^{n}_{(i-\frac{1}{2},k)}\right) - \frac{1}{24}\left(v_x{}^{n}_{(i+\frac{3}{2},k)} - v_x{}^{n}_{(i-\frac{3}{2},k)}\right)\right] + \frac{\Delta t\lambda_{(i,k)}}{\Delta h}\left[\frac{9}{8}\left(v_z{}^{n}_{(i,k+\frac{1}{2})} - v_z{}^{n}_{(i,k-\frac{1}{2})}\right) - \frac{1}{24}\left(v_z{}^{n}_{(i,k+\frac{3}{2})} - v_z{}^{n}_{(i,k-\frac{3}{2})}\right)\right]$$

$$\sigma_{zz}{}_{(i,k)}^{n+\frac{1}{2}} = \sigma_{zz}{}_{(i,k)}^{n-\frac{1}{2}} + \frac{\Delta t\left(\lambda_{(i,k)}+2\mu_{(i,k)}\right)}{\Delta h}\left[\frac{9}{8}\left(v_{z}{}_{(i,k+\frac{1}{2})}^{n} - v_{z}{}_{(i,k-\frac{1}{2})}^{n}\right) - \frac{1}{24}\left(v_{z}{}_{(i,k+\frac{3}{2})}^{n} - v_{z}{}_{(i,k-\frac{3}{2})}^{n}\right)\right] + \frac{\Delta t\lambda_{(i,k)}}{\Delta h}\left[\frac{9}{8}\left(v_{x}{}_{(i+\frac{1}{2},k)}^{n} - v_{x}{}_{(i-\frac{1}{2},k)}^{n}\right) - \frac{1}{24}\left(v_{x}{}_{(i+\frac{3}{2},k)}^{n} - v_{x}{}_{(i-\frac{3}{2},k)}^{n}\right)\right]$$

$$\sigma_{xz}{}_{(i+\frac{1}{2},k+\frac{1}{2})}^{n+\frac{1}{2}} = \sigma_{xz}{}_{(i+\frac{1}{2},k+\frac{1}{2})}^{n-\frac{1}{2}} + \frac{\Delta t\mu_{\left(i+\frac{1}{2},k+\frac{1}{2}\right)}}{\Delta h}\left[\frac{9}{8}\left(v_{x}{}_{(i+\frac{1}{2},k+1)}^{n} - v_{x}{}_{(i+\frac{1}{2},k)}^{n}\right) - \frac{1}{24}\left(v_{x}{}_{(i+\frac{1}{2},k+2)}^{n} - v_{x}{}_{(i+\frac{1}{2},k-1)}^{n}\right) + \frac{9}{8}\left(v_{z}{}_{(i+1,k+\frac{1}{2})}^{n} - v_{z}{}_{(i,k+\frac{1}{2})}^{n}\right) - \frac{1}{24}\left(v_{z}{}_{(i+2,k+\frac{1}{2})}^{n} - v_{z}{}_{(i-1,k+\frac{1}{2})}^{n}\right)\right]$$

Apéndice B. Discretización por diferencias finitas (FD) de segundo orden en tiempo y cuarto en espacio de las ecuaciones de campos adjuntos

$$
\begin{aligned}
{l_1}_{i+\frac{1}{2},k}^{m-\frac{1}{2}} = {l_1}_{i+\frac{1}{2},k}^{m+\frac{1}{2}} - \frac{\Delta t}{\rho_{i+\frac{1}{2},k}\Delta h}\Bigg[\frac{9}{8}\left[\left(\left(\lambda+2\mu\right)l_3\right)_{i+1,k}^{m} - \left(\left(\lambda+2\mu\right)l_3\right)_{i,k}^{m}\right] \\
-\frac{1}{24}\left[\left(\left(\lambda+2\mu\right)l_3\right)_{i+2,k}^{m} - \left(\left(\lambda+2\mu\right)l_3\right)_{i-1,k}^{m}\right]\Bigg] + \\
\left[\frac{9}{8}\left[\left(\lambda l_4\right)_{i+1,k}^{m} - \left(\lambda l_4\right)_{i,k}^{m}\right] - \frac{1}{24}\left[\left(\lambda l_4\right)_{i+2,k}^{m} - \left(\lambda l_4\right)_{i-1,k}^{m}\right]\right] + \\
\left[\frac{9}{8}\left[\left(\mu l_5\right)_{i+\frac{1}{2},k+\frac{1}{2}}^{m} - \left(\mu l_5\right)_{i+\frac{1}{2},k-\frac{1}{2}}^{m}\right] - \frac{1}{24}\left[\left(\mu l_5\right)_{i+\frac{1}{2},k+\frac{3}{2}}^{m} - \left(\mu l_5\right)_{i+\frac{1}{2},k-\frac{3}{2}}^{m}\right]\right]
\end{aligned}
$$

$$
\begin{aligned}
{l_2}_{i,k+\frac{1}{2}}^{m-\frac{1}{2}} = {l_2}_{i,k+\frac{1}{2}}^{m+\frac{1}{2}} - \frac{\Delta t}{\rho_{i,k+\frac{1}{2}}\Delta h}\Bigg(\left[\frac{9}{8}\left[\left(\lambda l_3\right)_{i,k+1}^{m} - \left(\lambda l_3\right)_{i,k}^{m}\right] - \frac{1}{24}\left[\left(\lambda l_3\right)_{i,k+2}^{m} - \left(\lambda l_3\right)_{i,k-1}^{m}\right]\right] + \\
\left[\frac{9}{8}\left[\left(\left(\lambda+2\mu\right)l_4\right)_{i,k+1}^{m} - \left(\left(\lambda+2\mu\right)l_4\right)_{i,k}^{m}\right] - \frac{1}{24}\left[\left(\left(\lambda+2\mu\right)l_4\right)_{i,k+2}^{m} - \left(\left(\lambda+2\mu\right)l_4\right)_{i,k-1}^{m}\right]\right] + \\
\left[\frac{9}{8}\left[\left(\mu l_5\right)_{i+\frac{1}{2},k+\frac{1}{2}}^{m} - \left(\mu l_5\right)_{i-\frac{1}{2},k+\frac{1}{2}}^{m}\right] - \frac{1}{24}\left[\left(\mu l_5\right)_{i+\frac{3}{2},k+\frac{1}{2}}^{m} - \left(\mu l_5\right)_{i-\frac{3}{2},k+\frac{1}{2}}^{m}\right]\right]\Bigg)
\end{aligned}
$$

$$
{l_3}_{i,k}^{m} = {l_3}_{i,k}^{m+1} - \frac{\Delta t}{\Delta h}\left[\frac{9}{8}\left[\left(l_1\right)_{i+\frac{1}{2},k}^{m+\frac{1}{2}} - \left(l_1\right)_{i-\frac{1}{2},k}^{m+\frac{1}{2}}\right] - \frac{1}{24}\left[\left(l_1\right)_{i+\frac{3}{2},k}^{m+\frac{1}{2}} - \left(l_1\right)_{i-\frac{3}{2},k}^{m+\frac{1}{2}}\right]\right]
$$

$$l_4{}_{i,k}^{m} = l_4{}_{i,k}^{m+1} - \frac{\Delta t}{\Delta h}\left[\frac{9}{8}\left[(l_2)_{i,k+\frac{1}{2}}^{m+\frac{1}{2}} - (l_2)_{i,k-\frac{1}{2}}^{m+\frac{1}{2}}\right] - \frac{1}{24}\left[(l_2)_{i,k+\frac{3}{2}}^{m+\frac{1}{2}} - (l_2)_{i,k-\frac{3}{2}}^{m+\frac{1}{2}}\right]\right]$$

$$\begin{aligned} l_5{}_{i+\frac{1}{2},k+\frac{1}{2}}^{m} = l_5{}_{i+\frac{1}{2},k+\frac{1}{2}}^{m+1} &- \frac{\Delta t}{\Delta h}\left[\frac{9}{8}\left[(l_1)_{i+\frac{1}{2},k+1}^{m+\frac{1}{2}} - (l_1)_{i+\frac{1}{2},k}^{m+\frac{1}{2}}\right] - \frac{1}{24}\left[(l_1)_{i+\frac{1}{2},k+2}^{m+\frac{1}{2}} - (l_1)_{i+\frac{1}{2},k-1}^{m+\frac{1}{2}}\right]\right] \\ &+ \left[\frac{9}{8}\left[(l_2)_{i+1,k+\frac{1}{2}}^{m+\frac{1}{2}} - (l_2)_{i,k+\frac{1}{2}}^{m+\frac{1}{2}}\right] - \frac{1}{24}\left[(l_2)_{i+2,k+\frac{1}{2}}^{m+\frac{1}{2}} - (l_2)_{i-1,k+\frac{1}{2}}^{m+\frac{1}{2}}\right]\right] \end{aligned}$$

Apéndice C. Pseudocódigo para la implementación de FWI elástica

Set model, discretization and CPML parameters
dV_{xtrue} , dV_{ztrue} ← READ TRUE SEISMOGRAMS ()
ρ, μ, λ ← READ INITIAL MODEL (ρ, V_p, V_s)
for $k = 0$ to $FWI\ iterations$ **do**
  for $it = 0$ to $Sources$ **do**
    for $it = 0$ to t_{sim} **do**
      V_x , V_z ← EXECUTE VELOCITIES KERNELS ($\sigma_{xx}, \sigma_{zz}, \sigma_{xz}, \rho, \mu, \lambda$)
      V_x , V_z ← EXECUTE VELOCITIES CPML KERNELS ($\sigma_{xx}, \sigma_{zz}, \sigma_{xz}, \rho, \mu, \lambda$)
      σ_{xx} , σ_{zz} ← EXECUTE SOURCE INJECTION KERNEL (S_x, S_z)
      σ_{xx} , σ_{zz} , σ_{xz} ← EXECUTE STRESSES KERNEL ($V_x, V_z, \rho, \mu, \lambda$)
      σ_{xx} , σ_{zz} , σ_{xz} ← EXECUTE STRESSES CPML KERNEL ($V_x, V_z, \rho, \mu, \lambda$)
      V_x^{cube} , V_z^{cube} ← SAVE VELOCITIES (V_x, V_z)
      dV_{xmod} , dV_{zmod} ← SAVE MODELED SEISMOGRAMS (V_x, V_z)
    end for

    CF ← CALCULATE COST FUNCTION(dV_{xmod} , dV_{zmod} , dV_{xtrue} , dV_{ztrue})

    for $it = t_{sim}final$ to 0 **do**
      l_1 , l_2 ← EXECUTE MISFITS INJECTION KERNEL ($dV_{xmod}, dV_{zmod}, dV_{xtrue}, dV_{ztrue}$)
      l_3 , l_4 , l_5 ← EXECUTE ADJOINT-STRESS KERNEL ($l_1, l_2, \rho, \mu, \lambda$)
      l_3 , l_4 , l_5 ← EXECUTE ADJOINT-STRESS CPML KERNEL ($l_1, l_2, \rho, \mu, \lambda$)
      l_1 , l_2 ← EXECUTE ADJOINT-VELOCITY KERNEL ($l_3, l_4, l_5, \rho, \mu, \lambda$)
      l_1 , l_2 ← EXECUTE ADJOINT-VELOCITY CPML KERNEL ($l_3, l_4, l_5, \rho, \mu, \lambda$)
      ψ_ρ , ψ_μ , ψ_λ ← EXECUTE GRADIENT CALCULATION $\left(V_x^{cube}, V_z^{cube}, l_1, l_2, l_3, l_4, l_5\right)$
    end for
  end for
  $\rho_k = \rho_{(k-1)} - \alpha_\rho \psi_{\rho_k}$
  $\mu_k = \mu_{(k-1)} - \alpha_\mu \psi_{\mu_k}$
  $\lambda_k = \lambda_{(k-1)} - \alpha_\lambda \psi_{\lambda_k}$
end for
ρ, V_p, V_s ← WRITE FINAL MODEL (ρ, λ, μ)