Desarrollo de un módulo para ProMAX® que realice el proceso de Inversión de Onda Completa (*FWI*) en tiempo para datos sísmicos adquiridos usando geometría *blended*

David Antonio Fuentes Vargas

Trabajo de grado para optar al título de Ingeniero Electrónico

Director

Ana Beatriz Ramírez Silva

PhD. en Ingeniería Eléctrica

Co-director

Julián Gonzalo Mantilla Arias

Magister en Ingeniería Electrónica

Universidad Industrial de Santander

Facultad de Ingenierías Fisico-Mecánicas

Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2017

A Dios por dirigir mi camino en cada meta que me he propuesto permitiéndome aprender algo nuevo que me haga crecer como persona, esta tesis es una de ellas.

A mis padres por acompañarme en este proceso, sus consejos, su cariño y su apoyo incondicional fueron claves para que lograra ser profesional,

a mi nono Luis y mi nona Teodosia que espero desde el cielo me esten mirando con una sonrisa,

a mi tía Alix y mi tío Fernando por estar constantemente pendientes de mí en el transcurso de la carrera,

a Karet por todo su amor y apoyo

y a todos aquellos que me han acompañado durante este proceso.

Agradecimientos

Este trabajo es apoyado por la empresa ECOPETROL S.A. y COLCIENCIAS como parte del proyecto 0266 de 2013. Agradezco a los profesores Ana Ramírez y Sergio Abreo por su tiempo, excelente disposición y por amablemente guiarme cuando no sabía como avanzar, también agradezco a Johan, Katherine, Jheyston, Julian, Fabian Noriega, David, Ivan, Fabian Sanchez y todos los integrantes del grupo CPS por apoyarme, escucharme y estar dispuestos a ayudarme a resolver las dudas y baches que surgieron en el transcurso de esta tesis.

Tabla de contenido

Lista de figuras

Lista de tablas

Resumen

Título: Desarrollo de un módulo para ProMAX® que realice el proceso de inversión de onda completa (*FWI*) en tiempo para datos sísmicos adquiridos usando geometría *blended*[*]

Autor: David Antonio Fuentes Vargas[**]

Palabras claves: Módulos en SeisSpace® ProMAX®, Inversión de onda completa, adquisición *blended*, *super-shot*

Descripción:

La inversión de onda completa es un método iterativo que estima modelos de velocidades del subsuelo con alta resolución. El modelo de velocidades es un punto de partida en posteriores etapas de procesamiento de datos sísmicos. La mayoría de las petroleras colombianas utilizan el *software* SeisSpace® ProMAX® para el procesamiento, y aunque conocen la existencia de la inversión de onda completa no la pueden utilizar en algún módulo de este *software*, ya que no se encuentra incluida.

Este trabajo implementa un módulo en SeisSpace® ProMAX® que internamente genera una adquisición utilizando geometría *blended* y aplica la inversión de onda completa a dichos datos, desarrollando ambos procesos en el lenguaje de programación CUDA-C. Una adquisición consiste en generar una perturbación en el subsuelo, la energía reflejada por las capas de la tierra es registrada por sensores en la superficie; después de cumplir un tiempo de espera, una nueva perturbación puede ser generada y se repite el proceso. La adquisición *blended* busca disminuir los tiempos de espera cambiando el disparo tradicional por un *super-shot*, un arreglo de múltiples disparos aplicados usando tanto posiciones como retardos en tiempo aleatorios. Se mejora la implementación de la inversión de onda completa propuesta por Flórez (2016) disminuyendo el tiempo de cómputo al aplicar cada *super-shot* sin utilizar el principio de superposición. Los modelos de salida obtenidos al usar una geometría *blended* y una tradicional son comparados en tiempo de cómputo y nivel de similitud con el modelo original.

[*] Trabajo de Grado modalidad en investigación

[**] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: PhD. Ana Beatriz Ramírez Silva. Codirector: Ms. Julián G. Mantilla Arias.

Abstract

Title: Development of a module for ProMAX® to perform the full waveform inversion (FWI) in time for seismic data acquired using a blended geometry[*]

Author: David Antonio Fuentes Vargas[**]

Keywords: Modules in SeisSpace® ProMAX®, Full Waveform Inversion, blended acquisition, super-shot.

Description:

The full waveform inversion is an iterative method that estimates high-resolution velocity models of the subsoil. The velocity model is a starting point for subsequent seismic data processing steps. Most of Colombian oil companies use SeisSpace® ProMAX® to the processing, and although they know the existence of the full waveform inversion they can't use it in any module of this *software*, since it isn't included.

This work implements a module in SeisSpace® ProMAX® that internally generates an acquisition using blended geometry and applies full waveform inversion to that data, both processes are developed in CUDA-C programming language. An acquisition consists of generating a disturbance in the subsoil, the energy reflected by the layers of the earth is recorded by sensors on the surface; a new disturbance can be generates after a timeout and repeat the process. Blended acquisition seeks to decrease the waiting times. This method changes the traditional shot by a super-shot, a set of shots applied using random time delays and random positions delays. Moreover, this work improves the implementation of the full waveform inversion proposed in by Flórez (2016) decreasing the computation time when applying each super-shot without using the superposition principle. The output models when using a combined geometry and a traditional geometry are compared in computation time and level of similarity with the original model.

[*]Degree Project

[**]Faculty of Physics Mechanics Engineering. Electrical, Electronics Engineering and Telecommunications School. Director: PhD. Ana Beatriz Ramírez Silva. Codirector: Ms. Julián G. Mantilla Arias.

Introducción

Los métodos de adquisición de hidrocarburos utilizados en una exploración sísmica se modifican dependiendo de la zona. La geometría de adquisición sísmica *blended* es una opción utilizada en la sísmica marina, que comparada con la adquisición con geometría uniforme permite registrar más información del subsuelo por disparo, disminuye el tiempo de cómputo y los tiempos de espera entre un disparo y el siguiente. La variante en este método de adquisición está en cambiar los disparos uniformes o tradicionales por conjuntos de disparos denominados *super-shots*. Entre las etapas de procesamiento sísmico, se encuentra el proceso de inversión sísmica, el cual permite obtener parámetros físicos del subsuelo como la velocidad, la densidad, etc. Para obtener una estimación del modelo de velocidades se aplica la inversión de onda completa (*FWI*, por sus siglas en inglés) a los datos registrados con la geometría de adquisición *blended*. La *FWI* parte de un modelo inicial y lo actualiza resolviendo la ecuación de onda acústica hasta cumplir un criterio de parada.

El grupo de investigación CPS[1] ha desarrollado un algoritmo computacional que aplica la *FWI* a una geometría de adquisición sísmica *blended* implementado en MATLAB. Sin embargo, surge la necesidad de incluir este algoritmo en los flujos de procesamiento utilizados en la industria. SeisSpace® ProMAX® es el *software* de procesamiento sísmico que la mayoría de empresas petroleras utilizan en Colombia y será la plataforma en la cual se

[1] Grupo de investigación en Conectividad y Procesamiento de Señales, Universidad Industrial de Santander

acople el algoritmo. Cabe recalcar que el grupo de investigación ya cuenta con un módulo en SeisSpace® ProMAX®[2] para realizar la *FWI*, pero aplicando geometría tradicional.

El algoritmo desarrollado en MATLAB no puede ser acoplado a ProMAX® debido a que no está programado en un lenguaje permitido (ProMAX® acepta el lenguaje C y FORTRAN). Además, la implementación debe permitir el uso de múltiples *GPUs*, a través del uso de lenguaje de programación en paralelo (CUDA-C).

Antes de acoplar el algoritmo a los módulos de trabajo de ProMAX® se buscan mejoras en el rendimiento computacional en comparación con el código suministrado. Durante el desarrollo del algoritmo en CUDA-C se encontró que el algoritmo en MATLAB hace uso del principio de superposición, resuelve la ecuación de onda para cada disparo de forma independiente y genera el *super-shot* sumando las trazas obtenidas. En el presente trabajo no se sigue este procedimiento, al considerarlo ineficiente en tiempo pues se debe lanzar varias veces un *kernel* de la *GPU* para obtener un *super-shot*. Se propone utilizar el *kernel* solo una vez para propagar todos los disparos que conforman el *super-shot*. También se modifican las fronteras absorbentes del propagador ya que su implementación en CUDA-C podría generar divergencia de *threads* dentro del *kernel*. Por otro lado, se aprovecha el uso de un lenguaje de programación en paralelo para resolver las ecuaciones diferenciales con diferencias finitas en un orden superior al de MATLAB, con la intención de obtener resultados más precisos.

[2] A partir de esta sección se usara ProMAX® para hacer referencia a SeisSpace® ProMAX®

1. Objetivos

1.1. Objetivo general

Integrar el algoritmo de inversión de onda completa (*FWI*), a un módulo de ProMAX® para obtener un modelo de velocidades para datos sísmicos adquiridos usando geometría *blended*.

1.2. Objetivos específicos

- Implementar un código que genere una geometría de adquisición *blended* en un módulo en ProMAX®.
- Modificar el módulo de ProMAX® de la *FWI* acústica con densidad constante 2D para que utilice datos sísmicos adquiridos con geometría *blended*.
- Comparar los modelos de velocidades obtenidos con la *FWI* aplicada a datos sísmicos adquiridos con geometría *blended* en un módulo de ProMAX® con su correspondiente resultado en MATLAB, evaluando el error cuadrático medio**.**
- Evaluar el tiempo de ejecución en ProMAX y compararlo con el rendimiento obtenido en MATLAB.

2. Exploración sísmica marina

Una adquisición sísmica marina (*off-shore*) parte de la generación de ondas de presión u ondas elásticas (con pistolas de aire o *air-gun*) dirigidas a la profundidad del subsuelo, una parte de la energía penetra en la tierra y es reflejada en las diferentes capas del subsuelo o estratos, al regresar a la superficie es registrada por una serie de sensores llamados hidrófonos, dispositivos que miden la intensidad y el tiempo que le toma a la onda reflejada regresar al sensor. La geometría más utilizada por la industria de exploración marina es la *Towed Streamer,* consiste en mantener hidrófonos en la superficie del agua (adheridos a un *Streamer* o cable) arrastrados por un buque de exploración (ver figura 2.1). La adquisición sísmica terrestre (*on-shore*) difiere en detalles operacionales pero maneja el mismo principio (Oil & Gas Producers [OGP], 2011).

La adquisición sísmica marina de disparos simultáneos (también llamada adquisición *blended*) es una forma económica de muestrear y acelerar el registro sísmico (Kumar, 2015). Una fuente tipo *blended* recibe el nombre de *super-shot*, y consta de la superposición de múltiples disparos (ver figura 2.2).

El procesamiento de datos sísmicos adquiridos con geometría de adquisición sísmica *blended* se puede realizar separando los disparos de cada *super-shot* y procesándolos de forma tradicional (a esta técnica se le denomina *deblending*) o procesando los datos mezclados. El enfoque utilizado en el algoritmo desarrollado por Flórez (2016) (con el cual

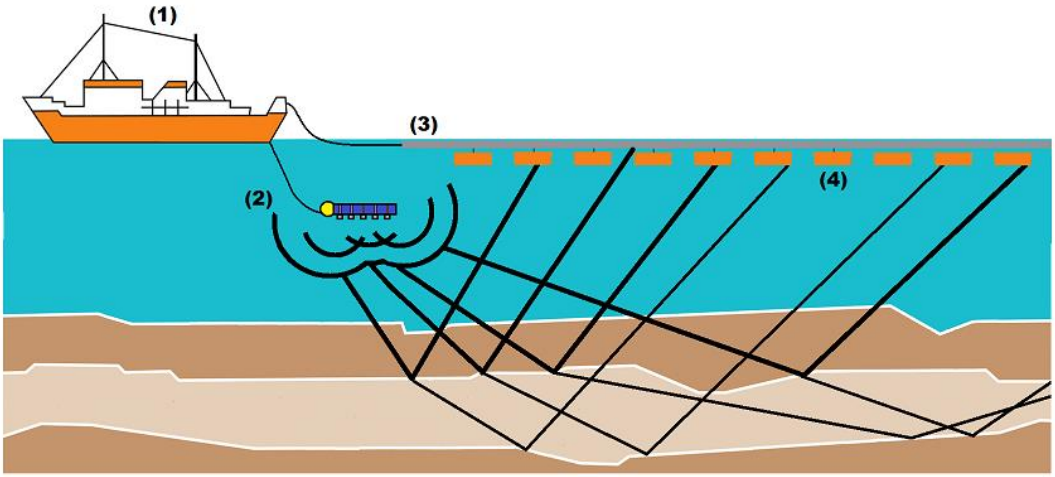

Figura 2.1 Towed Streamer **(1) Buque de exploración, (2) Fuente, (3) *Streamer,* (4) Flotador o *bird.* Nota: Adaptado de Flórez K., Inversión de Onda Completa (*FWI*) en tiempo para datos sísmicos adquiridos usando una geometría *blended*, 2016.**

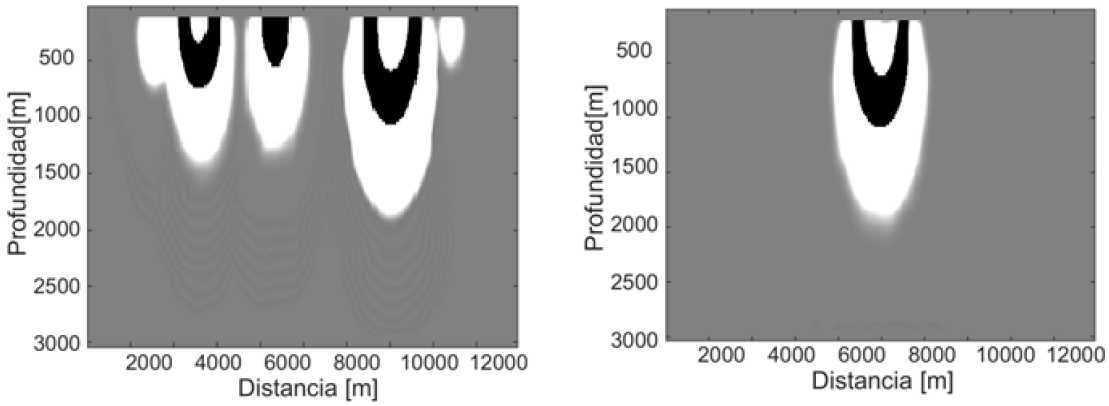

Figura 2.2 **Propagación de una adquisición *blended* (izquierda) y una adquisición tradicional (derecha).**

se comparan los resultados) que se toma como base para el desarrollo del presente trabajo es el procesamiento con datos mezclados, esta elección se fundamenta en que ofrece una ventajosa reducción en el tiempo de procesamiento proporcional al número de disparos por *super-shot* (Flórez, 2016).

En una adquisición tradicional, las fuentes son activadas con largos intervalos de tiempo para disminuir las interferencias entre disparos que puedan ser captadas por los receptores, lo cual conlleva a registros largos y costosos. Teóricamente el tiempo de espera entre dos disparos debe ser infinito, sin embargo, en la práctica ronda los 30[s]. El concepto de fuentes

simultaneas o adquisición *blended* aborda este problema reduciendo los tiempos de espera al realizar varios disparos en uno, disminuyendo los costos de exploración (Kumar, 2015).

La aplicación de fuentes simultaneas implica una interferencia que mal manejada puede afectar las posteriores etapas de procesamiento, como la *FWI*. Este fenómeno se reduce utilizando fuentes codificadas aleatoriamente en fase o tiempo (Flórez, 2016). En este trabajo se usan posiciones y retardos en tiempo aleatorios para contrarrestar la interferencia entre disparos, tal y como se maneja en el algoritmo implementado por Flórez (2016).

El algoritmo de la *FWI* necesita modelar la propagación de onda de presión en el medio. Para este propósito, se usa el operador (Bekhout, 2008)

$$F\{.\} = \left\{ \frac{1}{\boldsymbol{m}^2(x,z)} \frac{\partial^2 \boldsymbol{p}(x,z,t)}{\partial t^2} = \frac{\partial^2 \boldsymbol{p}(x,z,t)}{\partial x^2} + \frac{\partial^2 \boldsymbol{p}(x,z,t)}{\partial z^2} + src(x_0,t_0) \right\}, \qquad \textbf{(2.1)}$$

el cual describe el campo de onda en superficie para el caso de densidad constante con fuentes tradicionales, donde $src(x_0,t_0)$ es la fuente puntual, x_0 la posición espacial, t_0 el retardo en tiempo de la fuente, $\boldsymbol{m}$ es el campo de velocidad acústica en función de la posición y $\boldsymbol{p}$ es el campo de presión.

El propagador planteado en el operador (2.1) se utiliza para replicar la expansión de la onda acústica, que idealmente viaja hasta el infinito. No obstante, el modelo en el cual se propaga es finito, generando reflexiones no naturales con las fronteras. Para contrarrestar este fenómeno es necesario definir condiciones de frontera absorbentes que minimicen estas reflexiones indeseadas. Se han propuesto varios métodos para incluir dichas condiciones a la ecuación de onda, entre ellos *Absorbing Boundary Condition (ABC), Perfectly Matched Layer* (*PML*) y *Convolutional Perfectly Matched Layer* (*CPML*). Para incluir las condiciones de frontera *CPML* en la ecuación se requieren dos variables auxiliares (ψ, ζ) (Abreo, 2017).

Basándose en lo propuesto por Pasalic D. & Ray M. (2010) se modifica la ecuación de onda solucionada por el operador (2.1) para incluir *CPML*, obteniendo la ecuación

$$F\{.\} = \left\{ \frac{1}{\boldsymbol{m}^2(x,z)} \frac{\partial^2 \boldsymbol{p}(x,z,t)}{\partial t^2} = \frac{\partial^2 \boldsymbol{p}(x,z,t)}{\partial x^2} + \frac{\partial^2 \boldsymbol{p}(x,z,t)}{\partial z^2} + \partial\psi(x,z) + \zeta(x,z) \; + src(x_0,t_0) \right\}, \tag{2.2}$$

donde

$$\partial\psi(x,z) = \frac{\partial\psi(x,z)}{\partial x} + \frac{\partial\psi(x,z)}{\partial z}, \tag{2.3}$$

$$\zeta(x,z) = \zeta_x(x,z) + \zeta_z(x,z). \tag{2.4}$$

Las dos variables auxiliares $\psi(x,z)$ y $\zeta(x,z)$ son propuestas por Pasalic D. & Ray M. (2010) y serán definidas en el capítulo 4. La ecuación (2.2) será el nuevo operador que se utilizará para resolver la ecuación de onda.

El campo de onda de presión en la superficie está dado por $\boldsymbol{p}(x - x_0, z = 0, t - t_0)$, solución del operador (2.2) con diferencias finitas. Aplicando el principio de superposición, el dato adquirido *blended* en superficie se forma haciendo la suma lineal de los N disparos que se desea que maneje el *super-shot* con codificación de retardo en tiempo t_n y posición x_n (Florez, 2016), obteniendo la ecuación

$$\boldsymbol{p}_{bl} = \sum_{n=1}^{N} \boldsymbol{p}(x - x_n, z, t - t_n), \tag{2.5}$$

donde x_n y t_n son los retardos en posición y tiempo para cada una de las fuentes, $\boldsymbol{p}_{bl} \in \mathbb{R}^{N_x \text{ x } N_t}$ es el campo de onda en superficie adquirido con datos *blended* y $\boldsymbol{p} \in \mathbb{R}^{N_x \text{ x } N_t}$ el campo de onda tradicional en superficie.

3. *Full Waveform Inversion* (*FWI*)

La *FWI* es un algoritmo que estima parámetros en un modelo del subsuelo de forma detallada como la velocidad de propagación de onda. Para su funcionamiento necesita datos recogidos en campo y un modelo inicial. Éste último puede ser obtenido por procesos de tomografía, métodos de optimización global o usando información *a priori* (Serrano J., 2017). Los datos recogidos en campo reciben el nombre de datos observados. En este trabajo se utilizarán datos sintéticos simulados de la siguiente manera: como modelo inicial se toma una versión suavizada del modelo al cual se desea llegar después de aplicar la *FWI*, a este último se le denomina modelo original; los datos observados se obtienen propagando los *super-shots* en el modelo original para emular una adquisición en un subsuelo real.

3.1. Método del gradiente descendiente

El algoritmo de la *FWI* en cada iteración va modelando el campo de velocidades del subsuelo, para medir el grado de semejanza entre el dato que se está generando (dato modelado $\boldsymbol{d_{mod}}$) y el dato que se adquiere del subsuelo (dato observado $\boldsymbol{d_{obs}}$). La función de costo es el error dado por la norma l_2 (Jimenez,2016)

$$\phi(\boldsymbol{m}) = \frac{1}{2}\|\boldsymbol{d_{mod}}(\boldsymbol{m}) - \boldsymbol{d_{obs}}\|_2^2 , \qquad \textbf{(3.1)}$$

donde $\boldsymbol{d_{obs}}$ representa los datos observados y $\boldsymbol{d_{mod}}$ los datos modelados usando un modelo de velocidades $\boldsymbol{m}$, de dimensiones N_x por N_z.

La actualización del modelo de velocidades se realiza de forma iterativa con el método de Newton (Abreo, 2015)

$$\boldsymbol{m}_{k+1} = \boldsymbol{m}_k + \alpha_k h(\boldsymbol{m}_k)\,, \qquad \textbf{(3.2)}$$

donde $h(\boldsymbol{m}_k)$ es la dirección de búsqueda evaluada en $\boldsymbol{m}_k$, que puede ser determinada por el método de gradiente descendiente (*steepest-descent*) (Plessix R., 2006) o el L-BFGS (llamado así por sus desarrolladores Broyden, Fletcher, Goldfarb y Shanno) (Dos santos A. & Pestana R., 2015). Por otra parte, α_k es el tamaño de actualización en la dirección del gradiente, evaluada en la iteración k (Abreo, 2017).

La función del gradiente puede ser calculada con el método del estado adjunto (Plessix, 2006)

$$g(\boldsymbol{m}_k) = -\frac{2}{\boldsymbol{m}_k^3}\int_0^T \boldsymbol{\lambda}\frac{\partial^2 \boldsymbol{p}_s(x,z,t)}{\partial t^2}\,dt\,, \qquad \textbf{(3.3)}$$

donde T es el máximo tiempo de grabación, $\boldsymbol{p}_s(x,z,t)$ es el campo obtenido a partir de la propagación de fuentes puntales y $\boldsymbol{\lambda}$ es el campo obtenido de realizar la retropropagación del residual. La retropropagación consiste en aplicar el operador (2.2) al modelo usando como fuente la resta entre los datos modelados y observados, en el tiempo inverso.

Una vez que se encuentra el gradiente con el método del estado adjunto se actualiza el modelo de velocidades utilizando el método de gradiente descendiente (Abreo, 2017)

$$h_k = -g(\boldsymbol{m}_k). \qquad \textbf{(3.4)}$$

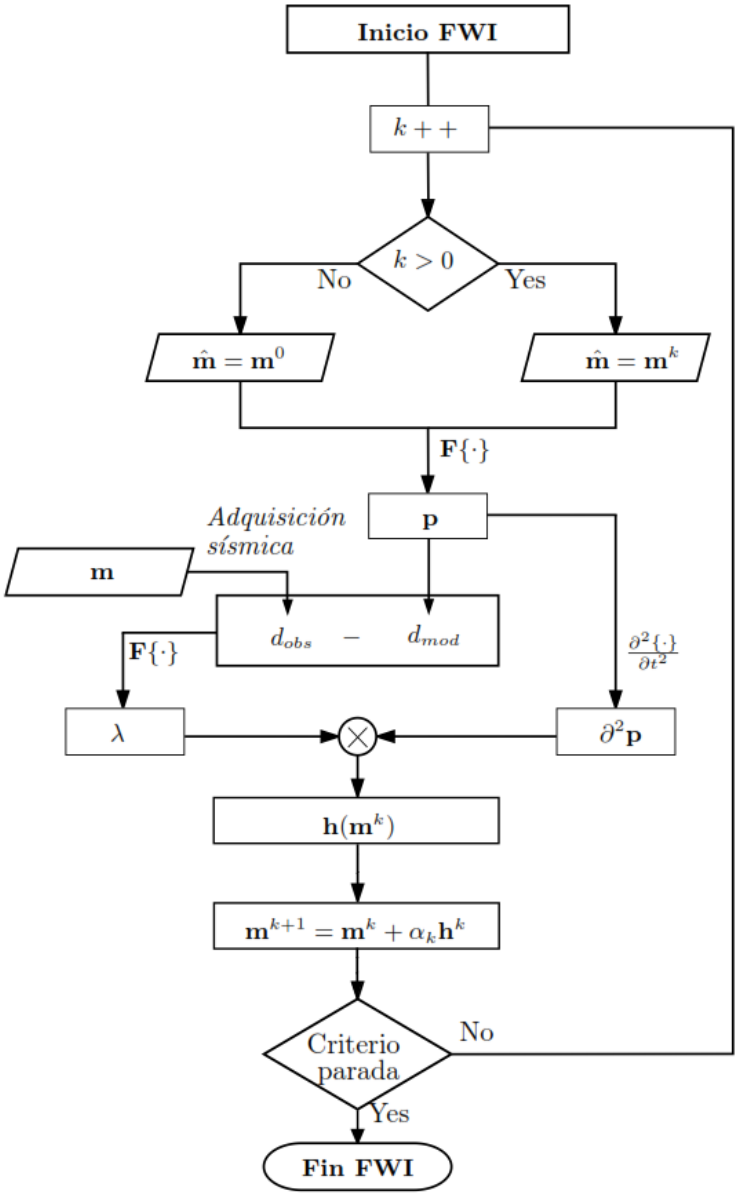

Figura 3.1 **Esquema general de una implementación *FWI*. Nota: Adaptado de Abreo D., Ramírez A. & Abreo S. "A hybrid methodology for a 3D full waveform inversion in time domain using GPUs". *15th International Congress of the Brazilian Geophysical Society EXPOGEf*. 2017.**

Para llegar a una relación de actualización dependiente del gradiente, se remplaza la relación (3.4) en (3.2), obteniendo

$$\boldsymbol{m}_{k+1} = \boldsymbol{m}_k - \alpha_k g(\boldsymbol{m}_k). \qquad \textbf{(3.5)}$$

En la figura 3.1 se representa por medio de un diagrama de flujo la teoría anteriormente expuesta.

3.2. Método L-BFGS

La *FWI* busca disminuir la función de costo para encontrar un dato modelado de alta resolución lo más fielmente posible al dato adquirido. El dato modelado se obtiene en cada iteración propagando sobre el modelo de velocidad que se va actualizando. El paso de actualización del modelo en cada iteración (conformado por el producto de $h(m_k)$ y α_k) es una parte clave del algoritmo, que al ser optimizada acelera la búsqueda del mejor modelo, reduciendo el tiempo de cómputo.

El método L-BFGS es una buena alternativa para encontrar una tasa de cambio del modelo en (3.2) que disminuya la función de costo más rápido. El algoritmo requiere el almacenamiento del gradiente y el modelo generado en las últimas m iteraciones (siendo $m \leq 10$), como se describe en el algoritmo 1, donde $\boldsymbol{s}_k = \boldsymbol{m}_{k+1} - \boldsymbol{m}_k$, $\boldsymbol{y}_k = G(\boldsymbol{m}_{k+1}) - G(\boldsymbol{m}_k)$ y $\sigma_k = 1/\boldsymbol{y}_k^T \boldsymbol{s}_k$. La matriz $\boldsymbol{D}_k^0$ puede ser aproximada por $\boldsymbol{D}_k^0 = \gamma_k \boldsymbol{I}$ con

$$\gamma_k = \frac{\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1}}{\boldsymbol{y}_{k-1}^T \boldsymbol{y}_{k-1}}, \tag{3.6}$$

este factor γ_k sirve como un operador de escalamiento de la matriz de dirección de búsqueda $\boldsymbol{h}_k$, que es equivalente a $-\boldsymbol{r}$ (Dos Santos, A. & Pestana, R., 2015; Flórez, 2016).

Luego de que el algoritmo 1 escala la dirección de búsqueda $\boldsymbol{h}_k$, se determina el paso de actualización del modelo ajustando $\alpha_k = 1$. El algoritmo realiza un escalamiento adecuado cuando se cumple la condición (Dos Santos, A. & Pestana, R., 2015).

$$\boldsymbol{J}(\boldsymbol{m}_{k+1}) < \boldsymbol{J}(\boldsymbol{m}_k), \tag{3.7}$$

donde $\boldsymbol{J}(\boldsymbol{m}_{k+1})$ es el valor de la función de error en la siguiente iteración al actualizar el

Algoritmo 1. Método L-BFGS

$\boldsymbol{q} \leftarrow G(\boldsymbol{m}_k)$;

for $i = k-1 : -1 : k-m$

$\varepsilon_i \leftarrow \sigma_i \boldsymbol{s}_i^T \boldsymbol{q}$;

$\boldsymbol{q} \leftarrow \boldsymbol{q} - \varepsilon_i \boldsymbol{y}_i$;

end for

$\boldsymbol{r} \leftarrow \boldsymbol{D}_k^0 \boldsymbol{q}$;

for $i = k-m : +1 : k-1$

$\beta \leftarrow \sigma_i \boldsymbol{y}_i^T \boldsymbol{r}$;

$\boldsymbol{r} \leftarrow \boldsymbol{s}_i(\varepsilon_i - \beta)$;

end for

---

modelo con el método L-BFGS y $\boldsymbol{J}(\boldsymbol{m}_k)$ es el valor de la función de error en la iteración presente. Si la condición (3.7) no se satisface la longitud de paso α_k se disminuye a la mitad y se repite el proceso hasta que la condición se cumpla.

Una estrategia de desarrollo de algoritmos computacionales es iniciar realizando pruebas en un modelo sencillo antes de pasar a los más complejos. En las primeras etapas del proyecto se diseñó una geometría *blended* realizando pruebas en un modelo sintético de velocidad constante (2000 [m/s]) y un cuadrado difractor en su centro con una velocidad superior (2500 [m/s]), las dimensiones del modelo son 121x497 con una distancia entre muestras de 25[m], que representan una sección de 3.025 [km] x 12.425 [km]. Se inició determinando la dirección de búsqueda para la *FWI* con el método del gradiente descendiente, posteriormente cuando se comprobó el correcto funcionamiento de la geometría se integró el método L-BFGS. Para mostrar la ventaja de usar L-BFGS se presenta en la figura *3.2* una comparación a 3 [Hz] del modelo de salida de ambos métodos después de

50 iteraciones y en la figura *3.3* tanto el modelo original como el cambio en la función de costo.

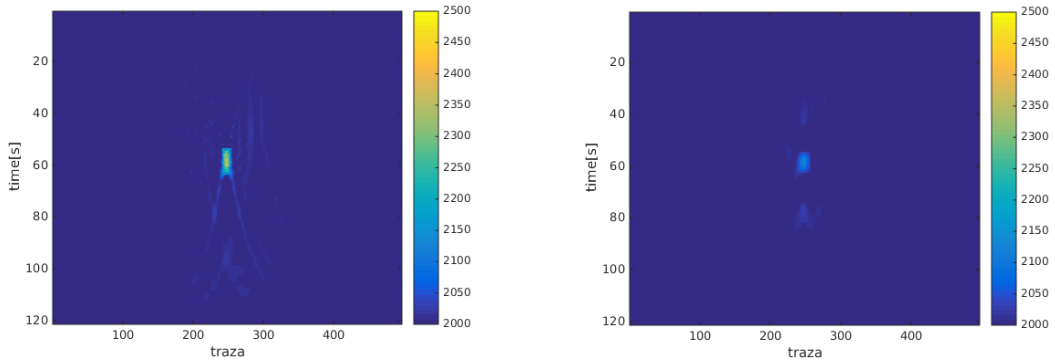

Figura 3.2 **Modelo de salida aplicando el método L-BFGS (izquierda) y gradiente descendiente (derecha).**

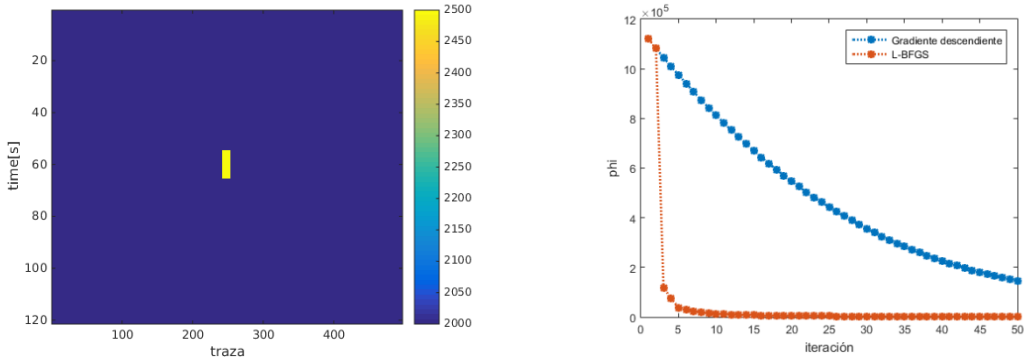

Figura 3.3 **Modelo original y Función de costo con los métodos L-BFGS y gradiente descendiente.**

Las funciones de costo en la figura 3.3 muestran que al aplicar el método L-BFGS a la *FWI* se reduce la función de costo más rápido: usando el método de gradiente descendiente se alcanza un valor de $1.445 * 10^5$ en la iteración 50 mientras que con el metodo L-BFGS se alcanza un valor de $1.173 * 10^5$ en tres iteraciones. Esta ventaja en la obtención de resultados en menos iteraciones es el argumento por el cual se aplica dicho método tanto en el código en MATLAB de Flórez (2016) como en el código en CUDA-C del presente trabajo. Cabe recalcar que usando el método de gradiente descendiente también se puede llegar a modelos de alta resolucion, pero en un tiempo de cómputo mayor.

4. Geometría de adquisición *blended*

La geometría de adquisición sísmica *blended* en este trabajo es diseñada con retardos de disparo entre 0.2 y 0.7 [s] con adquisiciones de 2.5 [s] como propone Flórez (2016). La codificación de cada *super-shot* contiene retardos aleatorios en tiempo t_n y posición x_n; estos valores son determinados con la función *rand()* en CUDA-C con una distribución de probabilidad uniforme, utilizando la fecha/hora del sistema como semilla.

El espaciamiento entre los disparos en cada *super-shot* debe manejar una restricción tanto en tiempo como en posición. El solo planteamiento de disparos distribuidos de forma aleatoria no es suficiente, se debe cumplir un espaciamiento entre disparos con la intención de evitar que algún área del modelo sea iluminada en mayor proporción que otra. La formula de espaciamiento propuesta por Flórez (2016) es:

$$esp = \frac{lim_{sup} - lim_{inf}}{2k}, \qquad \textbf{(4.1)}$$

donde k es el numero de disparos en cada *super-shot*. Cada disparo debe manejar dentro del *super-shot* una separación al menos $\pm esp$ del anterior (Flórez, 2016). La figura 4.1 presenta un *super-shot* de cinco disparos en las posiciones [371, 79, 207, 130, 421] y retardos temporales de [0.6789, 0.1946, 0.3912, 0.5802, 0.4934] respectivamente implementado en MATLAB con el propagador de Flórez (2016) y el mismo *super-shot* con el propagador tipo *blended* implementado en CUDA-C.

La diferencia de amplitud entre ambas implementaciones se puede atribuir a factores como el orden usado en la solución de las derivadas por diferencias finitas y el manejo de las fronteras absorbentes. Un *software* como MATLAB tiene la desventaja de tener que traducir los procesos a la CPU instrucción por instrucción, razón por la cual solucionar alguna de las

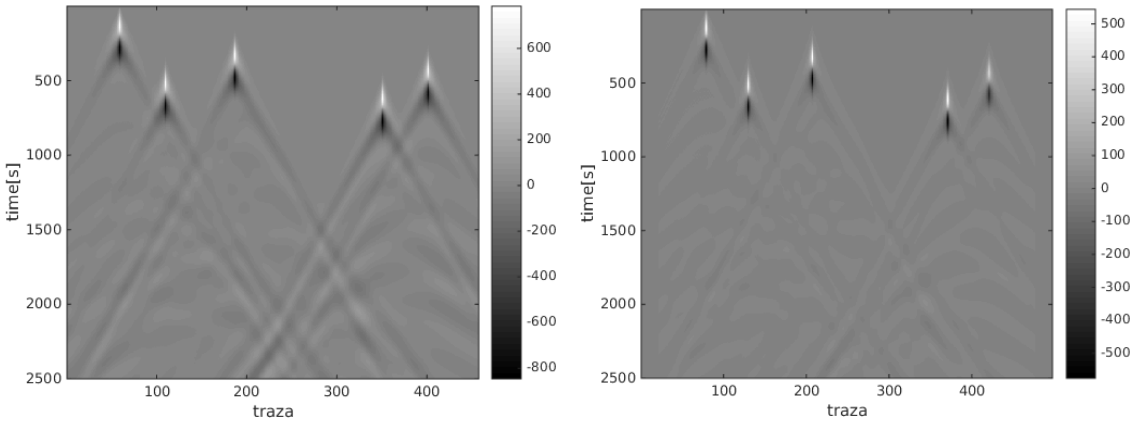

Figura 4.1 **Dato en superficie de un *super-shot* en MATLAB (izquierda) y CUDA-C (derecha).**

derivadas del propagador *blended* con diferencias finitas de un orden superior a dos es un proceso lento, en comparación con el tiempo que tardaría un *software* de lenguaje compilado. Un lenguaje como CUDA-C no tiene esta limitante, ya que acelera los procesos utilizando programación en paralelo sobre una *GPU*. Sin embargo, existe una disminución en la eficiencia de CUDA-C cuando maneja sentencias condicionales debido al *Thread Divergence* en los bloques del procesador.

4.1. Implementación en MATLAB (Flórez, 2016)

El operador $F\{.\}$ que representa la solución discreta de la ecuación de onda en (2.1) fue implementado por Flórez (2016) utilizando diferencias finitas centradas de segundo orden temporal y espacial, las fronteras absorbentes *CPML* se solucionan con diferencias finitas de primer orden. En la frontera derecha del modelo se utilizan diferencias finitas hacia atrás y tanto en la frontera izquierda como en la inferior diferencias finitas hacia adelante. La dirección en que se recorre las fronteras *CPML* en la solución de las derivadas va de afuera hacia adentro, es decir, la frontera *CPML* izquierda en la figura 4.2 se recorre en la dirección contraria a L_{XI}, la derecha en la dirección contraria de L_{XD} y la inferior en la dirección contraria de L_Z.

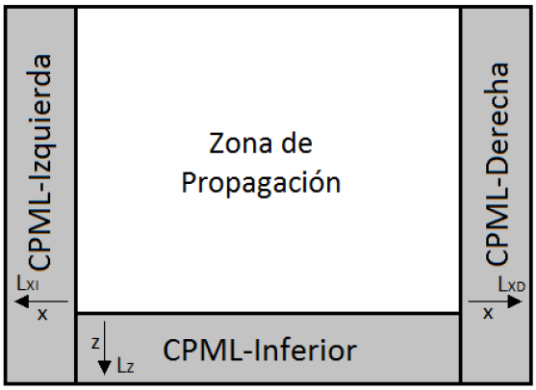

Figura 4.2 **Zona de propagación de onda incluyendo fronteras CPML. Nota: Adaptado de Perez C., Two-dimensional near-surface seismic imaging with surface waves: alternative methodology for waveform inversion. PhD thesis, *MINES ParisTech*, 2013.**

Una particularidad en la forma de plantear las fronteras absorbentes por Flórez (2016) está en el uso de atenuación con doble malla. En el capítulo 2 se presentó el operador (2.2) como la ecuación de onda acústica 2D teniendo en cuenta las fronteras absorbentes, definiendo $\psi(x,z)$ y $\zeta(x,z)$ como variables auxiliares del método propuesto por Pasalic D. & Ray M. (2010). Sin embargo, se hace necesario mostrar la obtención de estos valores con la intensión de argumentar la estrategia utilizada posteriormente en la implementación en CUDA-C.

Según Pasalic D. & Ray M. (2010), las variables que minimizan las reflexiones en las fronteras no naturales, vienen dadas por

$$\psi_q = b_q \psi_q + a_q \frac{\partial \boldsymbol{p}}{\partial q}, \tag{4.2}$$

$$\zeta_q = b_q \zeta_q + a_q \left(\frac{\partial^2 \boldsymbol{p}}{\partial q^2} + \frac{\partial \psi_q}{\partial q} \right), \tag{4.3}$$

donde q es la variable espacial del modelo que puede ser x o y, a_q y b_q determinan la atenuación de las variables auxiliares y $\boldsymbol{p}$ es el campo de presión resultante de la ecuación de onda.

Para encontrar los parámetros a_q y b_q se parte de los parámetros definidos por Perez (2013)

$$R = 0.001, \qquad L_q = att * \Delta q, \qquad d_0 = \frac{-3}{2L_q}\log(R).$$

Luego, los vectores F_x y F_z como

$$F_x = \begin{cases} L_x : \Delta x : 0 & \leftarrow x \in [0, att) \\ 0 & \leftarrow x \in [att, N_x - att) \\ 0 : \Delta x : L_x & \leftarrow x \in [N_x - att, N_x) \end{cases} \quad \textbf{(4.4)}$$

$$F_z = \begin{cases} 0 & \leftarrow z \in [0, N_z - att) \\ 0 : \Delta z : L_z & \leftarrow z \in [N_z - att, N_z) \end{cases}, \quad \textbf{(4.5)}$$

donde Δx y Δz son los pasos espaciales en dirección x y z, respectivamente; L_x y L_z son los límites de atenuación *CPML* en dirección x y z, respectivamente; att es la cantidad de puntos usados para realizar la atenuación, N_x es la cantidad de puntos del modelo en dirección de la coordenada x y N_z en la dirección de z (Abreo, 2017). Adicionalmente, Flórez (2016) define en el código en MATLAB otros dos vectores, generando una segunda malla de atenuación (Abreo, 2017):

$$F_{xh} = \begin{cases} L_x - \frac{\Delta x}{2} : \Delta x : 0 & \leftarrow x \in [0, att) \\ 0 & \leftarrow x \in [att, N_x - att) \\ 0 : \Delta x : L_x - \frac{\Delta x}{2} & \leftarrow x \in [N_x - att, N_x) \end{cases}, \quad \textbf{(4.6)}$$

$$F_{zh} = \begin{cases} 0 & \leftarrow z \in [0, N_z - att) \quad . \\ \\ 0 : \Delta z : L_z - \dfrac{\Delta z}{2} & \leftarrow z \in [N_z - att, N_z) \end{cases} \qquad \textbf{(4.7)}$$

Con los vectores (4.4) y (4.5) que recorren el modelo en las regiones *CPML* se calcula d_q y α_q usando F_q, así:

$$d_q = d_0 V_{max} \left(\frac{F_q}{L_q}\right)^2, \qquad \textbf{(4.8)}$$

$$\alpha_q = \pi f \left(\frac{L_q - F_q}{L_q}\right), \qquad \textbf{(4.9)}$$

donde V_{max} es la velocidad máxima del modelo y f es la frecuencia del *super-shot*. Sin embargo, la estrategia utilizada por Flórez (2016) para propagar también utiliza los vectores (4.6) y (4.7) como malla auxiliar, determinando adicionalmente unos valores d_{qh} y α_{qh} con las ecuaciones (4.8) y (4.9) respectivamente.

Finalmente se calculan b_q, a_q, b_{qh} y a_{qh} como (Abreo, 2017; Pasalic D. & Ray M., 2010)

$$b_q = e^{-(d_q + \alpha_q)dt}, \qquad \textbf{(4.10)}$$

$$a_q = \frac{d_q}{d_q + \alpha_q}\left(b_q - 1\right), \qquad \textbf{(4.11)}$$

$$b_{qh} = e^{-(d_{qh} + \alpha_{qh})dt}, \qquad \textbf{(4.12)}$$

$$a_{qh} = \frac{d_{qh}}{d_{qh} + \alpha_{qh}}\left(b_{qh} - 1\right). \qquad \textbf{(4.13)}$$

Los valores de atenuación encontrados de (4.10) a (4.13) son remplazados en las ecuaciones (4.8) y (4.9) dependiendo de la región a la que se aplica el método *CPML* de la figura 4.2. La correspondencia entre estas variables se presenta en la tabla 4.1.

Tabla 4.1 Variables auxiliares ψ_qy ζ_q dependiendo de la región CPML.

Regiones CPML laterales							
ψ_x		ζ_x		ψ_z		ζ_z	
a_{xh}	b_{xh}	a_x	b_x	a_z	b_z	a_z	b_z
Región CPML inferior							
ψ_x		ζ_x		ψ_z		ζ_z	
a_x	b_x	a_x	b_x	a_{zh}	b_{zh}	a_z	b_z

Esta distribución muestra que en las regiones que son recorridas en la coordenada x se utiliza primero a_{xh} y b_{xh} para determinar ψ_x y luego a_x y b_x para determinar ζ_x. Debido a la dependencia existente entre ψ_x y ζ_x que se mostró en la ecuación (3.3), el hecho de determinar estos valores con mallas distintas (es decir, una recorriendo el vector F_{xh} y la otra F_x) hace que la atenuación sea más robusta al utilizar más puntos de la región *CPML*. En la siguiente sección, se mostrará que aunque esta estrategia maneja el barrido de atenuación de una forma más sofisticada, el uso de una única malla para calcular ψ_x y ζ_x alcanza a atenuar en las fronteras lo suficiente para obtener un modelo de alta resolución.

Flórez (2016) se aprovecha el principio de superposición de la ecuación (1.5) para generar los *super-shots* como la suma de disparos tradicionales con un desfase temporal. La mejora que presenta esta forma de aplicar la *FWI* con geometría *blended* está en la retropropagación. En una geometría tradicional trabajar cinco disparos implica el cálculo de cinco propagaciones, cinco retropropagaciones y la determinación de cinco gradientes, la ventaja de la geometría de adquisición sísmica *blended* aplicada por Flórez (2016) está en que con un solo *super-shot* de cinco disparos generados bajo el principio de superposición solo necesita realizar una retropropagación y determinar un gradiente. Para mejorar este planteamiento, en el presente trabajo se genera el *super-shot* en CUDA-C con una sola propagación. Realizar una única propagación por *super-shot* además de reducir el número de propagaciones

tradicionales que debe realizar el algoritmo es una metodología más apegada al comportamiento que puede tener la fuente en una adquisición marina real.

4.2. Implementación en CUDA-C

El modelo de programación en CUDA-C fue creado en el 2006 por NVIDIA para aplicaciones específicas en áreas como la dinámica de fluidos, procesamiento de datos sísmicos, etc. Utilizando las *Graphics Processing Units* (*GPUs*), núcleos simplificados que se encargan de ejecutar instrucciones o *threads* de manera concurrente. En una *GPU* varios núcleos ejecutan instrucciones en paralelo, razón por la cual no se puede utilizar la programación en serie estándar, siendo remplazada por la programación heterogénea que permite aprovechar la capacidad de realizar procesos en paralelo de la *GPU* y utiliza

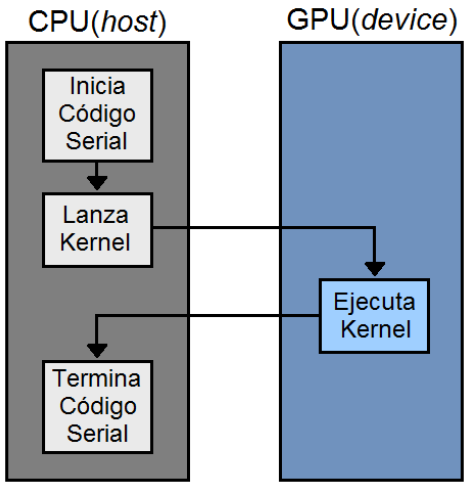

Figura 4.3 **Programación heterogénea. Nota: Adaptado de Suarez J., Comparación de las herramientas de programación en paralelo OpenCL y CUDA para la implementación de la propagación de la ecuación de onda acústica 3D con densidad constante basada en *STENCIL*, Universidad Industrial de Santander, 2016.**

también la CPU o *host* para instrucciones secuenciales. La CPU actúa como anfitrión lanzando *kernel*s a la *GPU* para las instrucciones que necesita procesar, como se presenta en la figura 4.3 (Suarez, 2016).

El modelo de programación en CUDA-C consta de tres componentes principales: el *Thread* o hilo el cual se encarga de ejecutar instrucciones a un dato o conjunto de datos en específico, el *Threads Blocks* es una agrupación de 1024 *threads* o menos y por último la *Grid* (Enmallado), la cual agrupa los *Threads Blocks*. Estos tres componentes se presentan en la figura 4.4 (Suarez, 2016).

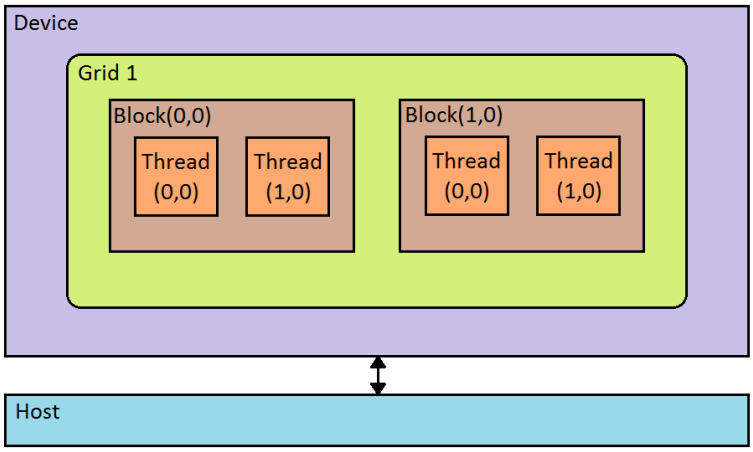

Figura 4.4 **Jerarquía en CUDA-C. Nota: Adaptado de Suarez J., "Comparación de las herramientas de programación en paralelo OpenCL y CUDA para la implementación de la propagación de la ecuación de onda acústica 3D con densidad constante basada en *STENCIL*," Universidad Industrial de Santander, 2016.**

Los *Threads* de un bloque se agrupan en *warps*. Los *Threads* dentro de un *warp* deben seguir la misma trayectoria de ejecución y todos los *Threads* deben ejecutar la misma instrucción al mismo tiempo. En otras palabras, los *Threads* no pueden divergir (Cornell University, 2017).

La sentencia más común que causa divergencia entre hilos o *Thread Divergence* es la ramificación por condicionales en una instrucción *if-then-else.* Si algunos *Threads* en un mismo *warp* se evalúan como verdaderos y otros como falsos, entonces los *Threads* verdaderos y los falsos se ramificarán en diferentes instrucciones. CUDA-C tiene una solución para este problema, pero tiene consecuencias negativas en el rendimiento. Al ejecutar una instrucción *if-then-else,* CUDA-C primero ejecutara el *then* y luego el *else.*

Mientras se ejecuta el *then* todos los *Threads* que se evalúan como falsos son desactivados. Cuando la ejecución cambia a la condición *else*, la situación se invierte. Como se puede ver esta instrucción no se ejecuta en paralelo si no en serie, generando una significativa perdida de rendimiento (Cornell University, 2017).

Aunque en la implementación propuesta por Flórez (2016) las fronteras *CPML* con doble malla generan una atenuación más densa que utilizando solo una, este método requiere remplazar b_q, a_q, b_{qh} y a_{qh} en ψ_q y ζ_q dependiendo de la región del modelo, lo cual implica usar condicionales en el *kernel*. También utiliza derivadas hacia adelante o hacia atrás dependiendo de la región que se esté manejando, agregando aún más condiciones en los *kernels* del código. Estas sentencias condicionales pueden generar un *Thread Divergence* desfavorable para el rendimiento del algoritmo pensando en una implementación en CUDA-C. Por lo tanto, pensando en reducir el costo computacional del algoritmo no se utiliza la doble malla para calcular ψ_q y ζ_q, se hallan con una única malla utilizando b_q y a_q. Además, las derivadas en CUDA-C son determinadas usando diferencias finitas centradas en las tres regiones que se desea atenuar, evitando la condición de derivar hacia adelante o hacia atrás dependiendo de la región.

En la sección 4.1 se habló del principio de superposición de la ecuación (2.5) que fue utilizado por Flórez (2016) para generar los *super-shots*. En la figura 4.5 se presenta la diferencia entre un *super-shot* realizado como la suma de disparos tradicionales y el *super-shot* generado con una única propagación propuesto en este trabajo. El rango de valores del *super-shot* va de 544.33 a -575.98 (ver figura 4.1), y como se aprecia en la figura 4.5 la magnitud de la diferencia esta aproximadamente cinco décadas por debajo de este rango.

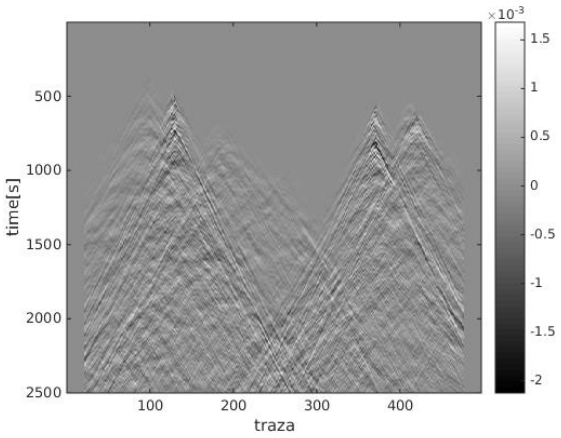

Figura 4.5 **Diferencia entre *super-shot* bajo el principio de superposición y el propuesto.**

El error cuadrático medio entre el *super-shot* que usa el principio de superposición sumando la propagación de varios disparos tradicionales y el *super-shot* que usa solo una propagación para generar los disparos es de 0.23. Un error lo suficientemente pequeño para comprobar que ambas formas de propagación son aproximadamente equivalentes.

5. Módulo *FWI* en ProMAX®

5.1. ProMAX®

ProMAX® es un *software* de procesamiento de datos sísmicos desarrollado por Landmark que tiene la opción de agregar nuevas herramientas de procesamiento o módulos personalizados a los ya existentes. Teniendo un código de *FWI* para geometrías de adquisición *blended*, se propone aprovechar esta opción que ofrece el *software* para crear un nuevo módulo que integre el algoritmo. Para agregar módulos sin que el desarrollador pueda afectar los flujos predeterminados, el *software* deja realizar estas acciones por fuera de la jerarquía interna, en una carpeta de usuario enlazada (Jiménez, 2016).

5.2. Creación del módulo en ProMAX®

Los algoritmos que se agregan a ProMAX® deben ser integrados en el entorno de SeisSpace® en un ambiente de programación de Fortran o C. El archivo *Makefile* permite compilar códigos por separado creando archivos objeto que luego pueden ser unidos en un único ejecutable, esta facilidad se aprovecha enlazando el archivo objeto en CUDA-C con el archivo objeto del código en C de ProMAX® que contiene el algoritmo de procesamiento. La interfaz que manejará el usuario no muestra códigos de programación, solo contiene los parámetros, entradas y salidas que necesita el módulo para realizar el procesamiento. Al código que enlaza la interfaz de usuario y el código del programa se le denomina **menú** (Jiménez, 2016).

5.2.1. Jerarquía de directorios

La jerarquía de directorios para desarrollo se crea utilizando el programa *Makeadvance*[3], que produce un espejo de la jerarquía de directorios de ProMAX® en la carpeta de usuario. De esta manera, se pueden hacer cambios en el *software* en un directorio personal sin ingresar directamente a su jerarquía interna (Monsegny, 2015).

Las variables de entorno de la jerarquía son definidas en el *shell bash* (~/.bashrc) de Linux. Dentro del cual se debe verificar que la variable de entorno PROMAX_HOME sea igual a la ruta del directorio donde está instalado ProMAX® y escribir los alias:

[3] *Makeadvance: shell script* en PROMAX_HOME/puerto/bin para crear jerarquías de directorios (Jiménez, 2016).

```
alias promake='/usr/bin/make
 -I$PROMAX_HOME/port/include/make'
alias promakedb='/usr/bin/make debug=yes
 -I$PROMAX_HOME/port/include/make'
alias Makeexec='$PROMAX_HOME/port/bin/Makeexec
 -I$PROMAX_HOME/port/include/make'
```

se cargan los cambios en el *bash* (~/.bashrc) por consola con el comando *source* y se genera el espejo de la variable de entorno PROMAX_HOME con *Makeadvance*:

```
>> source ~/.bashrc
>> Makeadvance
```

después de crear la jerarquía de directorios, se incluye una nueva variable de entorno en el *bash* (~/.bashrc) con la ruta de instalación de ProMAX®, en este trabajo la variable de entorno será nombrada $MY_PROMAX_HOME

```
export MY_PROMAX_HOME=~/”ruta de instalación del software”
```

en la figura 5.1 se muestra la jerarquía de directorios $MY_PROMAX_HOME que será utilizada para generar el módulo.

La creación del nuevo módulo parte de generar un directorio en la ubicación MY_PROMAX_HOME/port/src/exe, en el cual se guarda el código en C, el archivo *Makefile* y el código del algoritmo en CUDA-C.

5.2.2. Programa

El programa es el código que realiza el procesamiento de los datos con la información que ingresa el usuario a través del menú, lo hace ya sea por sí mismo o enlazando códigos externos, su estructura se presenta en el código 1.

El código en C tiene tres librerías que son *cglobal.h*, *cpromax.h* y *cSocketTool.h*. ProMAX® maneja variables globales que generalmente son asignadas a parámetros de

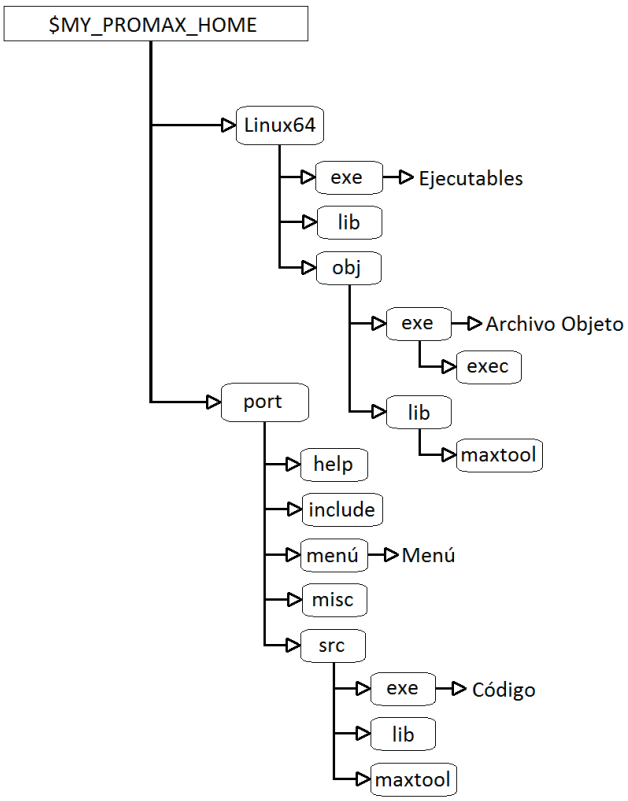

Figura 5.1 **Jerarquía de directorios de ProMAX®. Nota: Adaptado de Jiménez O., Desarrollo de un módulo para ProMAX que realice el proceso de inversión de onda completa (FWI) para la obtención de modelos de velocidades. Bucaramanga 2016.**

entrada del módulo, la librería que contiene dichas variables es *cglobal.h*. La librería *cpromax.h* contiene las librerías estándar de C y las funciones para crear tablas y bases de

Código 1. Estructura del código en C. Tomado de Monsegny (2015).

```
//Header ProMAX
#include <cglobal.h>
#include <cpromax.h>
#include <cSocketTool.h>

//Main
int main(int ac, char **av)
{
  initStandAlone(ac,av, "ST_NOOP");
  stConnectToServer();
  stSetToolName("No operation");

  //Initialization code goes here

  stEndInitialization ();
```

```
    //Processing code goes here using stGetTrace,
    //stGetEnsemble, stPutTrace and stPutEnsemble

    stCloseSocketLink();
    XtDisplay();
    return EXIT_SUCCESS;
}
```

datos. La tercera librería es *cSocketTool.h*, que permite con la función *stConnectToServer()* abrir la conexión con ProMAX® y con *stCloseSocketLink()* cerrarla (Jiménez, 2016).

Las funciones *initStandAlone()* y *stToolName()* se encargan de abrir y nombrar el programa que se desea añadir, respectivamente.

En "*Initialization code goes here"* se declaran las variables y se asigna los parámetros ingresados desde el menú. Después de *"stEndInitialization"* se ingresa el código que procesa los datos ingresados (Monsegny, 2015).

La compilación del programa requiere de un archivo *Makefile* que contenga las órdenes de ejecución.

5.2.3. Makefile

El propósito del *Makefile* es enlazar y compilar todos los archivos que conforman el módulo, uniendo el código base, funciones externas como las programadas en CUDA-C y el menú.

En su estructura general, se debe agregar tanto el nombre del archivo en C como del directorio en que se encuentra (Jiménez, 2016; Monsegny, 2015).

El archivo *Makefile* debe encontrarse en el mismo directorio del programa en C. Una vez se termine de hacer los cambios deben ser compilados usando el comando promake, en caso de no haber ningún error de programación los cambios serán cargados a ProMAX®.

5.2.4. Librerías de CUDA-C en ProMAX®

Las librerías de CUDA-C que necesita la *FWI* deben ser agregadas desde el archivo *Makefile*. Hay tres modificaciones que se deben hacer al *Makefile* para enlazar el código ***.cu** de la *FWI* con geometría *blended* a ProMAX®. En primer lugar se agrega el kit de herramientas de CUDA-C escribiendo la ubicación dentro del equipo (Jiménez, 2016)

```
LIBSCUDA += /usr/local/cuda-7.5/lib64/libcudart.so .
```

Posteriormente se agrega en la etapa de enlace de archivos objeto $(exe):$(uobjs)$(libs) el archivo ***.o,** y se enlazan las librerías de CUDA-C: $(exe):$(uobjs) FWIbld_v1.o $(libs) $(LIBSCUDA).

Por último, se añade la orden de compilación

```
FWIbld_v1.o: FWIbld_v1.cu
      nvcc FWIbld_v1.cu –c
```

5.2.5. Menú

La interfaz gráfica donde el usuario ingresa los parámetros de procesamiento es diseñada y controlada por el archivo ***.menu** que se crea en la ubicación MY_PROMAX_HOME/port/menu/promax/. Un menú en ProMAX® está conformado por cuatro partes: los cabeceros, la especificación de los parámetros, exec data y las reglas (ver código 2) (Monsegny, 2015).

Código 2. Esquema general del menú. Tomado de Monsegny (2015)

```
'(
   ;********Menu heading********
   name: PROGRAM_NAME
   label: "Sample ProMAX Menú"
   value_tab: 48
   ;**Parameter Specifications**
   parameter: PARM
      ; Attributes of the parameters
   ;*********Exec_Data**********
```

```
   exec_data: ("PROGRAM_NAME"
      ; Parameters of execution
          )
   ;***********Rules************
   rules: (
     ; Reglas
        )
)
```

Menu heading

Esta parte del menú consta de tres instrucciones: ***name***, ***label*** y ***value_tab***. La entrada ***name*** es opcional, y sirve para identificar el programa al que pertenece el menú; el nombre con que aparecerá el modulo en ProMAX® se ingresa en ***label*** y la instrucción ***value_tab*** sirve para definir el número de espacios de carácter que existirá entre el extremo derecho del menú y las casillas en las que el usuario podrá ingresar los parámetros (Monsegny, 2015).

Parameter specifications

Los parámetros ingresados por el usuario tienen su propio formato de declaración, a modo de ejemplo se presenta en el código 3 la declaración del número de *super-shots* que maneja el módulo.

Código 3. Ejemplo de declaración del parámetro NSS en el menú.

```
parameter: NSS
text: "Number of super-shots"
type: typein:
type_desc: ( int: 4 nil 1 )
value: 4
mouse_text: "Write the number of super-shots (between 1 and 4)"
```

En el ejemplo presentado en el código 3, *parameter* corresponde al nombre de la variable en el menú (nombre con el que se enlaza al código C), *text* es la cadena de caracteres que aparecerá en la interfaz de usuario requiriendo la entrada del dato, *type* describe la forma de

entrada del dato (por teclado, tabla, dataset, etc.), *type_desc* sirve para añadir especificaciones adicionales, para el ejemplo en particular del código 3 describe el ancho de la casilla de entrada, *value* es el valor predeterminado del parámetro y *mouse_text* el texto que describe el parámetro que se debe ingresar.

Exec data

Es la parte del menú en que se especifican las variables del menú y el archivo ***.exe** (Jiménez, 2016). A continuación se presenta un ejemplo simple de cómo se adiciona una variable y el ejecutable del programa en C:

```
("program" implicit: "fwibld.exe")
("NSS" implicit: (value 'NSS))
```

Donde *fwibld.exe* es el programa que se ejecuta y "*NSS"* la variable que almacena el dato que ingresa el usuario y es enlazada al código en C.

Rules

Las reglas controlan qué parámetros se pueden ocultar al usuario cuando no son necesarias. La sección de reglas se maneja con un condicional siguiendo el formato (Jiménez, 2016):

```
rules:
( rule_name (condition)
            (if it is true)
            (if it is false)
)
```

En el módulo del algoritmo de la *FWI* para datos *blended* se utilizan dos reglas para permitir al usuario elegir entre un procesamiento con una única frecuencia o con multiescala de frecuencias. Las condiciones utilizadas se presentan en el código 4.

Código 4. Ejemplo de reglas del menú.

```
rules:
```

```
(
      (rule1 ( = 1 ( value 'LOADER ))
            (do_show '(single_freq))
            (do_not_show '(single_freq))
      )
      (rule2 ( = 2 ( value 'LOADER ))
            (do_show '(NF first_freq second_freq third_freq))
            (do_not_show '(NF first_freq second_freq third_freq))
      )
)
```

La variable *LOADER* se ingresa al módulo siguiendo el formato explicado en las "*parameter specifications"* pero asignando una bandera de 1 o 2 si el usuario eligió una única frecuencia o un procesamiento multi-frecuencia, respectivamente. Siguiendo los condicionales, el código 4 muestra en la interfaz gráfica la posibilidad de ingresar una única frecuencia de trabajo si *LOADER* vale 1 o las NF frecuencias y sus valores correspondientes si *LOADER* vale 2.

5.2.6. Instalar el menú

Para que el menú aparezca en la lista de procesos de ProMAX® y pueda ser utilizado por el usuario es necesario crear un archivo de texto con nombre *ProcessesPM* en la misma ubicación del menú. Este nuevo archivo maneja la siguiente sintaxis:

Código 5. *ProcessesPM* del módulo

```
 '(
      ("Prototypes"
            ("Full Waveform Inversion Blended" "fwibld" h)
      )
 )
```

El texto "*Full Waveform Inversion Blended"* será el nombre con el que el usuario de ProMAX® podrá ver y reconocer el módulo entre los ya existentes. La asignación del menú se realiza agregando el nombre del archivo ***.menu**, para el módulo de este trabajo el nombre es fwibld.menu.

6. Resultados

El análisis de la herramienta desarrollada se realiza evaluando los resultados obtenidos en términos de rendimiento computacional e imágenes de velocidad del subsuelo. Para esto se utilizó el modelo Marmousi[4] de 121x497 puntos con una distancia entre muestras de 25[m], representando una sección de 3.025 [km] x 12.425 [km], como modelo original (ver figura 6.1).

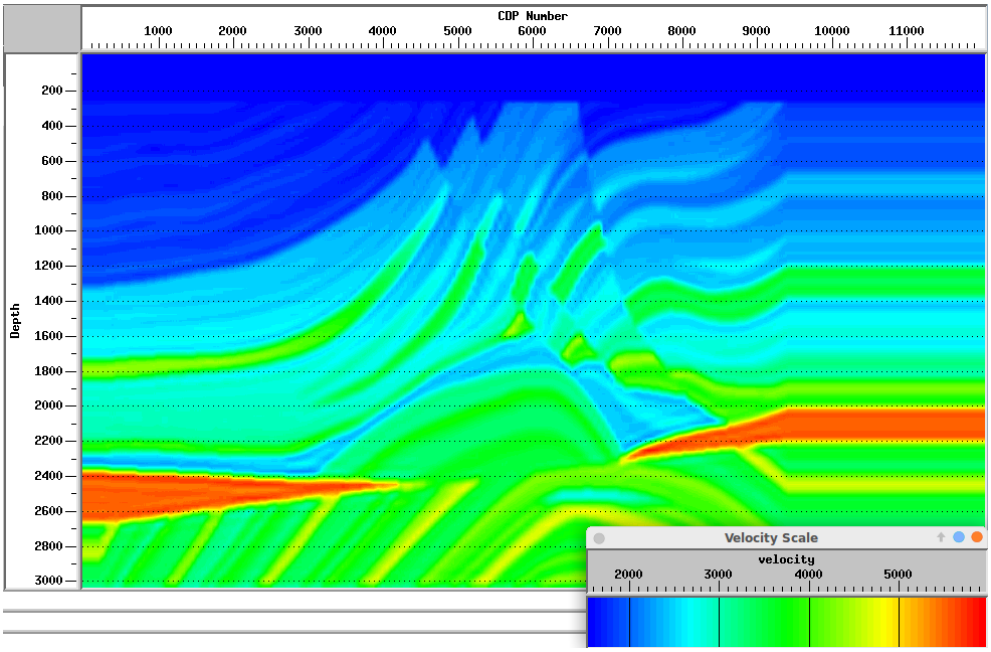

Figura 6.1 **Visualización en ProMAX® del modelo original.**

Cada pixel del modelo original es promediado 20 veces con una ventana de 5x5 pixeles colindantes, en el eje horizontal para obtener el modelo suavizado que se utiliza como modelo inicial (ver figura 6.2).

[4] Versteeg, R & Grau, G. (1991). *The Marmousi experience: Proceedings of the 1990 EAEG Workshop on practical aspects of seismic data inversion*.

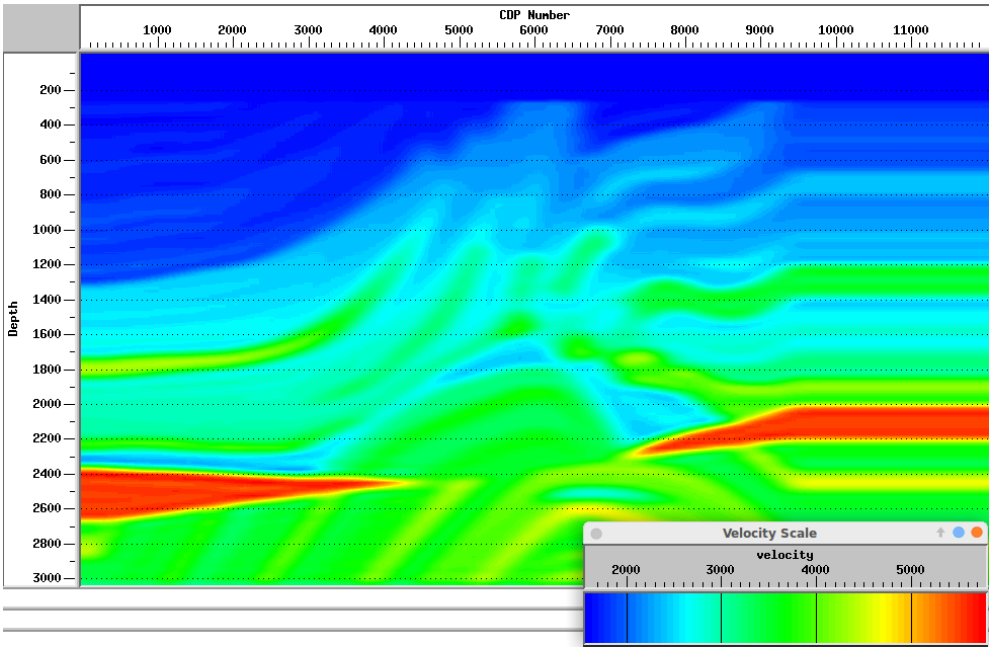

Figura 6.2 **Visualización en ProMAX® del modelo inicial.**

Para la codificación de las trazas, el algoritmo desarrollado calcula aleatoriamente retardos temporales entre 0.2 y 0.7 [s] y posiciones espaciales entre 20 y 477 puntos. Estos valores cambian cada vez que se ejecuta el código, razón por la cual se toma una muestra para las pruebas con posiciones espaciales [371, 79, 207, 130, 421] a una profundidad de 6 puntos (es decir, 150 [m]) y [0.6789, 0.1946, 0.3912, 0.5802, 0.4934] [s] como retardos temporales.

En la figura 6.3 se presenta el modelo obtenido con las especificaciones consignadas en la tabla 6.1 implementado por Flórez (2016) y en la figura 6.4 el modelo obtenido siguiendo estos mismos parámetros.

Tabla 6.1 Parámetros de implementación

Parámetro	Valor
Intervalo de punto en la grilla	20 [m]
Paso de tiempo	1 [ms]
Dimensiones X y Z del modelo	497x121
Número de *super-shots*	4
Número de *shots* por *super-shot*	5
Frecuencia	3 [Hz]
Número de iteraciones	50
Alpha inicial	9

El error cuadrático medio normalizado (ECMN) entre la implementación realizada en CUDA-C acoplada a ProMAX® comparada con la presentada por Flórez (2016) se calcula

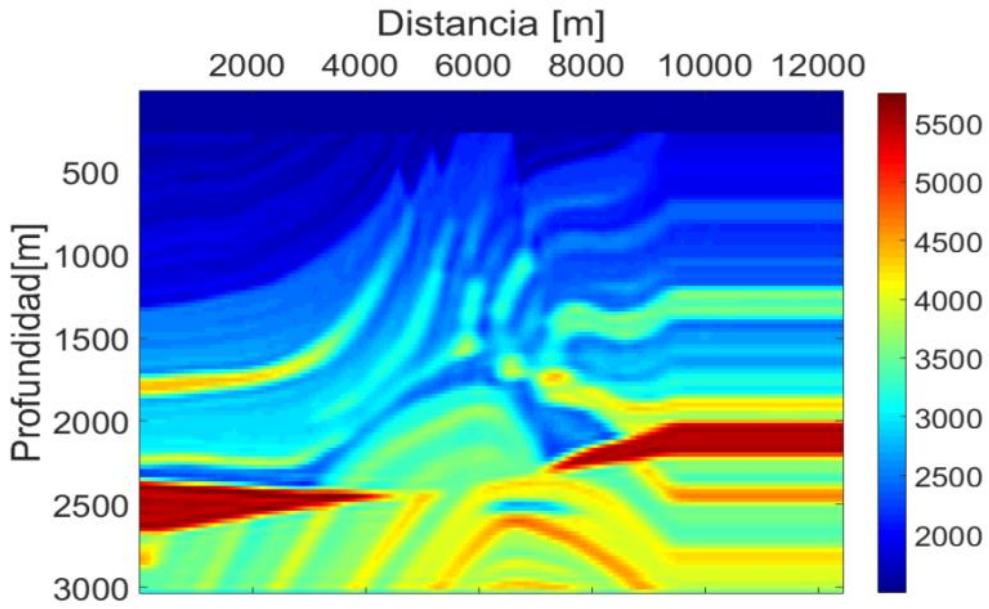

Figura 6.3 **Modelo de velocidades obtenido en MATLAB.**

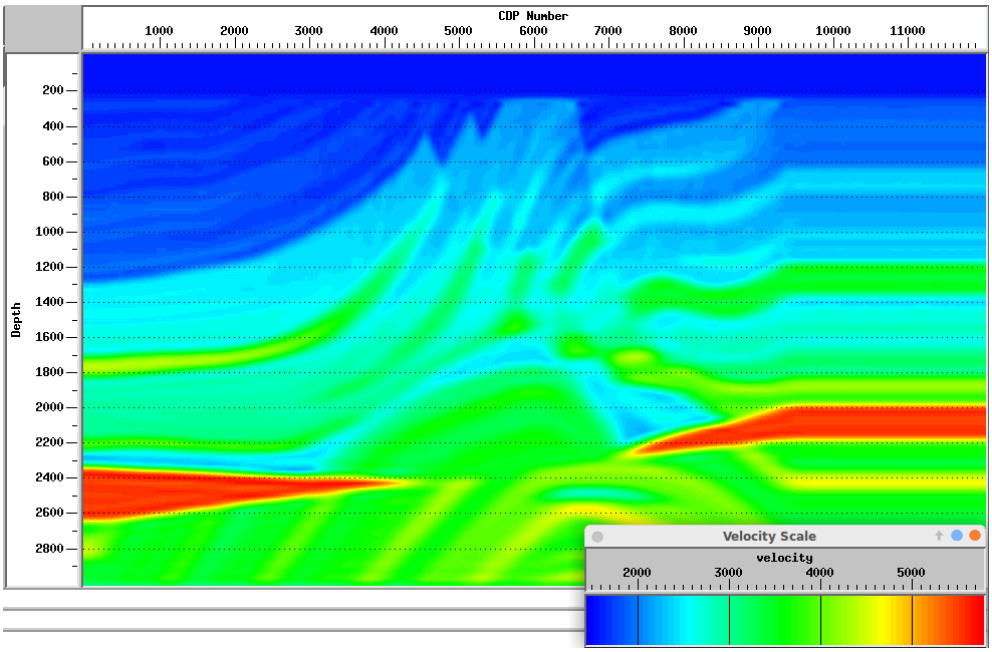

Figura 6.4 **Modelo de velocidades obtenido en ProMAX®.**

aplicando la ecuación 6.1, que dice

$$ECMN = \frac{\|\boldsymbol{m}_{vel}^{MATLAB} - \boldsymbol{m}_{vel}^{ProMAX}\|_2^2}{\|\boldsymbol{m}_{vel}^{MATLAB}\|_2^2}, \qquad \textbf{(6.1)}$$

donde $\boldsymbol{m}_{vel}^{MATLAB}$ es el modelo encontrado en MATLAB por Flórez (2016) y $\boldsymbol{m}_{vel}^{ProMAX}$ el modelo obtenido con el módulo implementado en ProMAX®. El calculo del numerador y denominador de la ecuacion 6.1 consiste en elevar al cuadrado las componentes de la matriz

que representa el modelo y sumar los resultados, es decir

$$\|\boldsymbol{m}_{vel}^{MATLAB}\|_2^2 = \sum_{i=1}^{N_x N_z} (\boldsymbol{m}_{vel,i}^{MATLAB})^2. \qquad \textbf{(6.2)}$$

El ECMN definido en la ecuación 6.1 entre los modelos hallados en ProMAX® y MATLAB es de $8.6 * 10^{-3}$ donde el tiempo de ejecución fue de 976 [s] (16 [min] 16 [s]) en ProMAX® y 75143[s] (20[h] 52[min] 22[s]) en MATLAB.

6.1. Comparación de las implementaciones

La función de costo presentada en la seccion 3.1 será comparada en ambas implementaciones con los valores normalizados (ver figura 6.5).

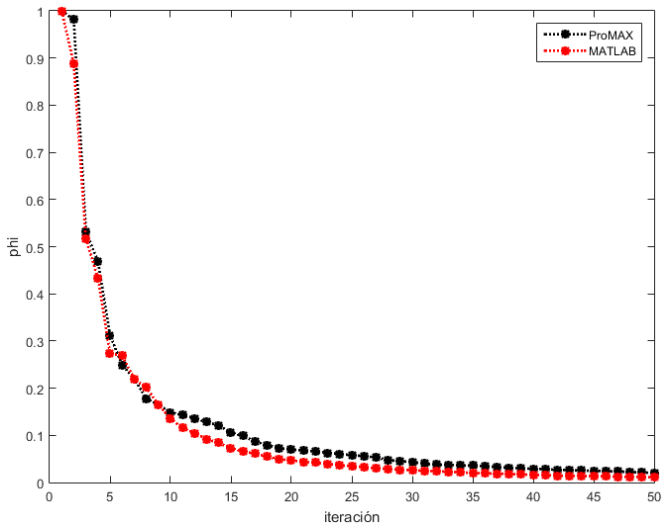

Figura 6.5 **Comparación de funciones de costo**

La función de costo es la evaluación del error con la norma l_2 en cada iteración. Se observa en la figura 6.5 que el error maneja una tendencia similar en ambas implementaciones, sin embargo, la evaluación de la función de costo no es una métrica que

permita conocer detalladamente las diferencias entre un modelo estimado y el original, debido a que un único parámetro no puede decir que partes específicas del modelo presentan mayor o menor similitud, solo mide el error entre todas las trazas como un parámetro global. Otra métrica que evalúa con un único parámetro si dos modelos son semejantes es el error cuadrático medio normalizado (ECMN). Al comparar los modelos estimados en MATLAB y en ProMAX® con el modelo original, se obtiene los valores presentados en la tabla 6.2, cuantificando el nivel de semejanza de los modelos. Esta métrica presenta la misma particularidad de la función de costo, solo engloba una semejanza general.

Tabla 6.2 Error cuadrático medio normalizado (ECMN) entre los modelos y el modelo original

Modelo de salida	ECMN con el modelo original
MATLAB	$38.9x10^{-3}$
ProMAX®	$36.9x10^{-3}$

Otra estrategia de comparación es la evaluación de un perfil 1D de los modelos de salida. En la figura 6.7 se presenta la comparación del perfil de velocidad obtenida con las implementaciones en MATLAB y ProMAX®, en 6750 [m].

La evaluación del modelo en este perfil 1D en particular permite observar ciertas regiones en las cuales la *FWI* que se realiza en ProMAX® es más cercana al modelo original. Por ejemplo, el modelo en ProMAX® presenta mayor semejanza comparado con el modelo en MATLAB a una profundidad entre 1400 [m] y 1600 [m].

La decisión de cual implementación genera mejores resultados o incluso el grado de semejanza entre dos resultados de *FWI* debe basarse en alguna métrica que englobe la gran mayoría de detalles del modelo. Una alternativa más general está en realizar un análisis de *Cycle Skipping*.

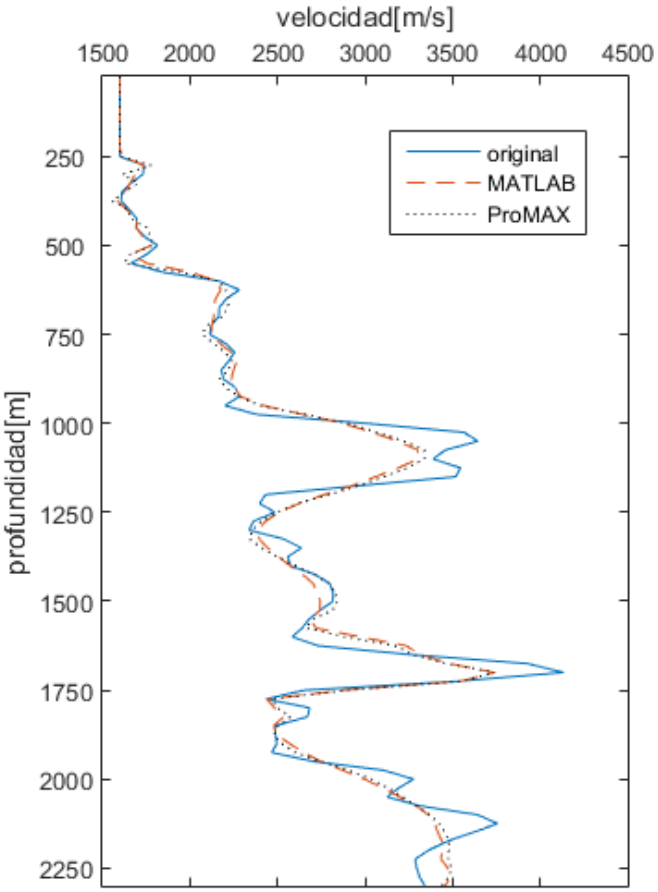

***Figura 6.6 1D profile* para cuatro *super-shots* de cinco disparos a 6750 [m].**

6.1.1. Análisis de *Cycle Skipping*

Para poder obtener un modelo adecuado, es decir cerca del vecindario del mínimo global en la función de costo, se deberá evitar caer en problemas de *cycle skipping* (CS). El CS ocurre cuando existe un desfase mayor que la mitad del periodo de la fuente (Yong, M., 2012). Por ejemplo, en la figura 6.7(a) se presenta un desfase mayor a $T/2$ con respecto a la figura 6.7 (b) es decir con CS. El lóbulo n de la figura 6.7 (a) se ajusta al lóbulo $n-1$ de la figura 6.7 (b) es decir con CS. Por otro lado la figura 6.7 (c) presenta un desfase menor a $T/2$ con respecto a la figura 6.7 (b) es decir sin CS. El lóbulo n de la figura 6.7 (c) se ajusta al lóbulo n de la figura 6.7 (b) (Yong, 2012).

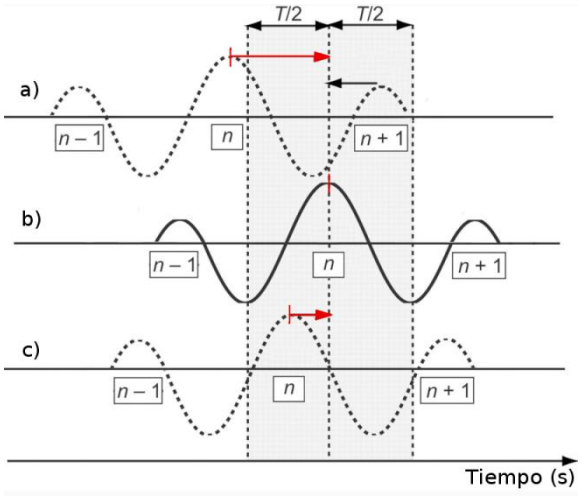

Figura 6.7 **Condición para *cycle skipping*. Adaptada de Yong M., "Waveform-based velocity estimation from reflection seismic data," Doctoral Thesis, Colorado School of Mines, 2012.**

El análisis de CS consiste en comparar las trazas modeladas con las trazas observadas restando sus fases y verificando si esta diferencia presenta CS al ser mayor a $T/2$. Las fases se determinan con la trasformada de Hilbert en todos los tiempos de cada traza. La figura 6.8 presenta la forma en que se puede comparar dos modelos evaluando el criterio de CS. El procedimiento consiste en: primero determinar las trazas observadas y modeladas (en el modelo inicial para este caso particular), tal y como se aprecia en la figura 6.8 (a) y 6.8 (b). Luego se determinan las fases instantáneas de cada traza con la transformada de Hilbert, obteniendo las figuras 6.8 (c) y 6.8 (d). En la figura 6.8 (e) se presenta la diferencia entre las fases instantáneas de ambos conjuntos de trazas. Como se explicó antes, el CS se presenta cuando la diferencia entre las fases es mayor a $T/2$, es decir a π. Las fases instantáneas que cumplen esta condición aportan un 1 en la figura 6.8 (f), generando un mapeo de puntos en 1 ó 0 que permite identificar las zonas con problemas de CS (Serrano J., Abreo S. & Ramírez A., 2017).

Finalmente, el porcentaje de CS se determina sumando los valores igual a 1 (*bits*) de la figura 6.8 (f) como

$$SC = \frac{\sum bits\ con\ SC}{Nt * Nx} * 100\ , \qquad \textbf{(6.3)}$$

donde Nt es el número de muestras temporales, Nx es el número de trazas y los $bits\ con\ SC$ son los condicionales en la figura 6.8 (f), que toman un valor de 1 si existe SC o de 0 si no (Serrano J., Abreo S. & Ramírez A., 2017).

El porcentaje encontrado de CS en la figura 6.8 solo es válido para ese *super-shot* en particular, una forma de determinar el porcentaje total de CS en el modelo consiste en sumar

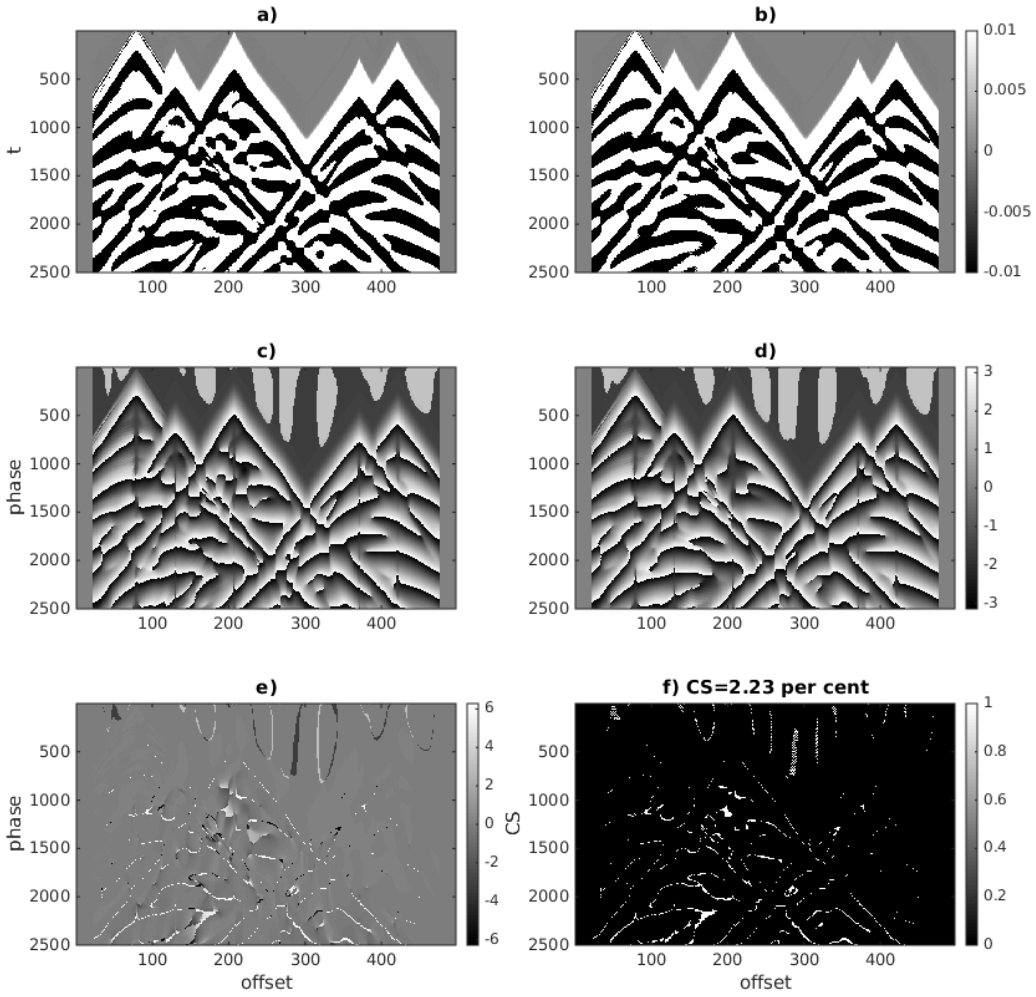

Figura 6.8 **Procedimiento de obtención de CS entre el modelo original e inicial. a) Trazas observadas, b) trazas modeladas, c) fase de las trazas observadas, d) fase de las trazas modeladas, e) resta entre las fases de las trazas observadas y modeladas y f) porcentaje de CS en el *super-shot* y mapeo de condicionantes con un 1 si hay CS o un 0 en caso contrario.**

los bits condicionales de cada traza en todos los *super-shot* y llegar a un perfil en dos dimensiones que presente el conteo de fases instantáneas con problemas de CS. A modo de ejemplo se muestra en la figura 6.9 el porcentaje total de CS en el modelo inicial.

Una ventaja de esta métrica es que permite detectar las trazas más problemáticas dentro de cada *super-shot*, convirtiéndose en una herramienta muy útil al momento de diseñar una geometría de adquisición sísmica *blended* con menor interferencia.

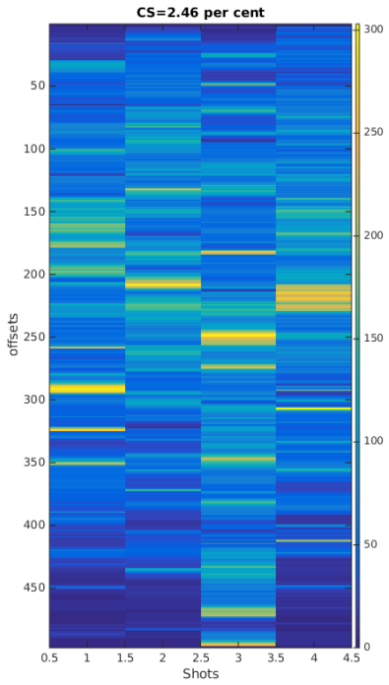

Figura 6.9 **CS total del modelo inicial en las trazas de cada super-shot**

La evaluación del CS explicada en las figuras 6.8 y 6.9 debe realizarse a los modelos de salida utilizando ambas implementaciones, debido a que son diferentes. Los resultados se observan en las figuras 6.10 y 6.11.

El análisis de CS presentado en las figuras 6.10 y 6.11 muestra la similitud entre las dos implementaciones, los valores de las diferencias de desfase son semejantes. El porcentaje de CS en las trazas adquiridas con ambos propagadores cambia ligeramente tanto usando el

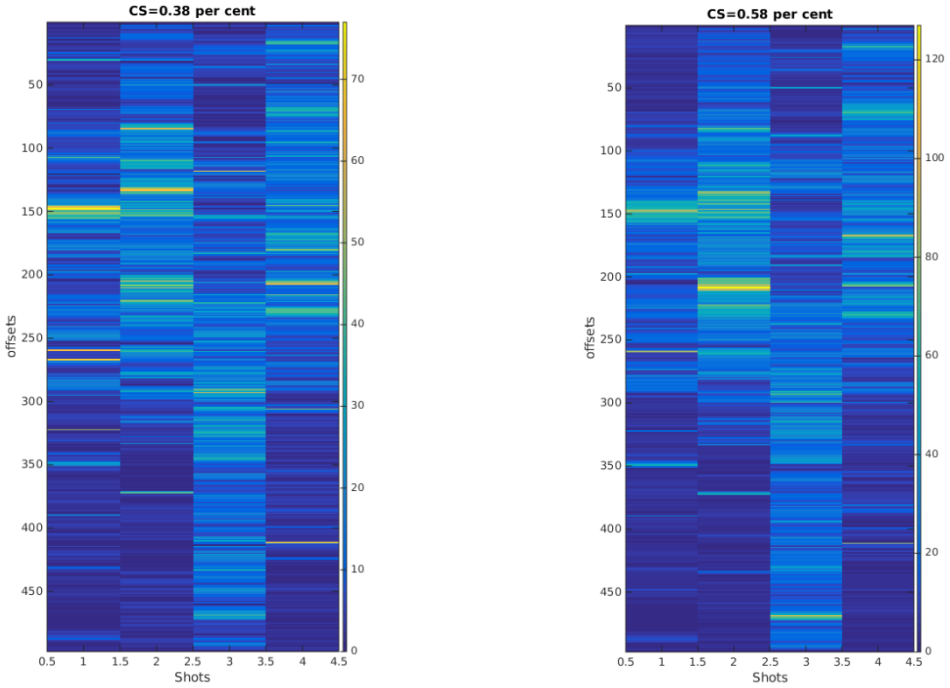

Figura 6.10 **CS total usando el propagador implementado en CUDA-C del modelo de salida obtenido en ProMAX® (izquierda) y MATLAB (derecha).**

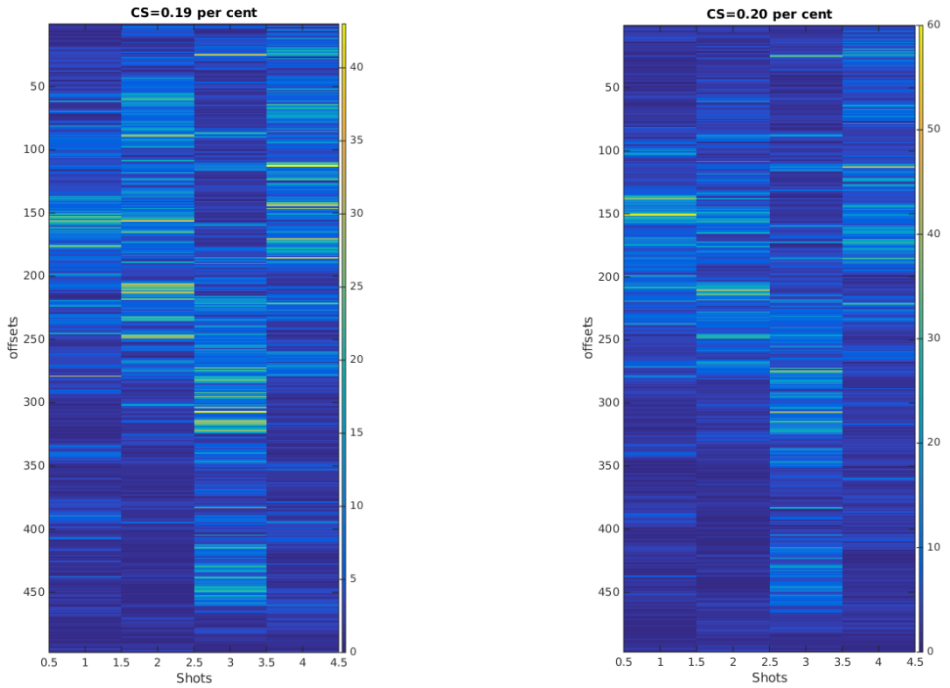

Figura 6.11 **CS total usando el propagador en MATLAB del modelo de salida obtenido en ProMAX® (izquierda) y MATLAB (derecha).**

propagador de ProMAX® (de 0.58% en el modelo de salida de MATLAB a 0.38% en el modelo de salida en ProMAX®) como usando el propagador en MATLAB (de 0.20% en el modelo de salida de MATLAB a 0.19% en el modelo de salida en ProMAX®). El

propagador implementado en CUDA-C y acoplado a ProMAX® muestra más detalles que no permite ver el propagador en MATLAB debido a que maneja un orden de diferencias finitas superior.

6.2. Módulo implementado en ProMAX®

La interfaz de usuario del módulo en ProMAX® implementado permite elegir entre aplicar la *FWI* a una geometría de adquisición *blended* generada internamente o añadir la geometría de forma externa. Los modelos original e inicial que se presentan en las figuras 6.1 y 6.2 respectivamente, son ingresados al módulo por medio de tablas de parámetros, las cuales almacenan los datos en el disco y se cargan en la memoria cuando son necesarios. Las tablas de parámetros son visualizadas con la herramienta *Velocity Viewer/Point Editor** de ProMAX®. La geometría adquirida en campo puede cargarse seleccionando la opción *Load* en el parámetro *Geometry Blended* del menú que se presenta en la figura 6.12.

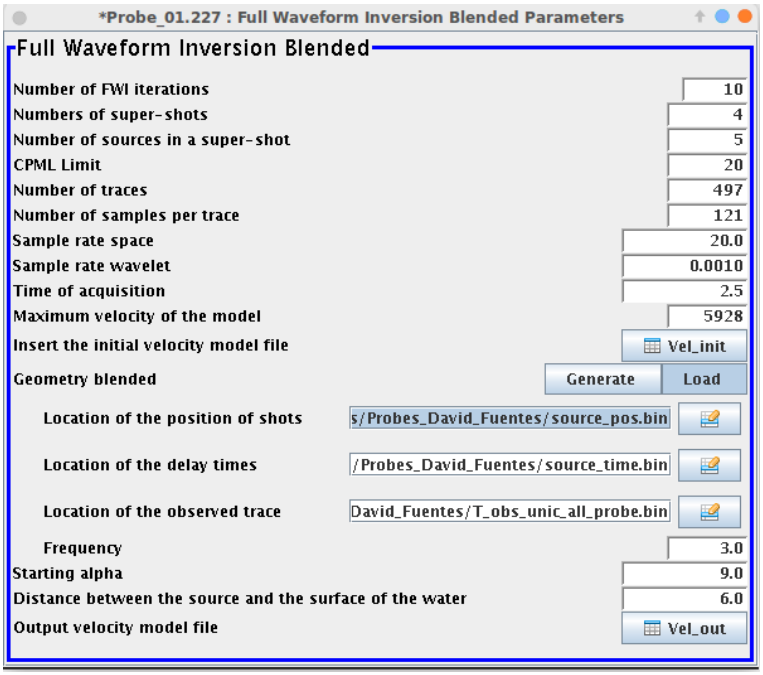

***Figura 6.12* Menú del módulo en ProMAX® seleccionando la opción *Load*.**

Al seleccionar la opción *Load* en el menú se despliegan cuatro parámetros, el primero es *Location of the position of shots* en el cual se debe añadir la ubicación de las posiciones de

los disparos contenidos en cada *supershot* dentro del equipo, guardadas como archivo binario. El parámetro *Location of the delay times* pide los retardos de tiempo de cada *shot*. El parámetro *Location of the observed traces* debe incluir la ubicación del dato adquirido en campo guardado como vector, que será interpretado internamente por el algoritmo. Finalmente, *Frequency* es la frecuencia a la cual fueron registrados los datos.

La opción *Load* permite cargar geometrías de adquisición *blended* externas, pero siguiendo el objetivo planteado en este trabajo de generar una geometría de adquisición sísmica *blended* en el módulo, también se tiene la opción *Generate* que se presenta tanto en la figura 6.13 como en la figura 6.14.

Para implementar una geometría de adquisición *blended* generada automáticamente por el módulo, se requiere del modelo original en el cual se realizarán las propagaciones *blended* para formar las trazas observadas. El módulo internamente crea posiciones y retardos en tiempo aleatorios espaciados cumpliendo la condición de la ecuación 4.1. En la figura 6.13 se presenta el módulo listo para realizar la *FWI* con la geometría de adquisicion sísmica *blended* generada internamente a una única frecuencia de 3 [Hz]. En la figura 6.14 se presenta una alternativa eligiendo la opción *multiscale*, para aplicar la *FWI* a varias escalas empezando por las frecuencias bajas. Este tipo de procesamiento es una forma de reducir el *cycle skipping* entre los datos observados y simulados.

El enfoque multiescala consiste en realizar la inversión en varias etapas a frecuencias distintas. La información de alta frecuencia de los datos tiene más posibilidades de presentar *cycle skipping* que la parte de baja frecuencia debido a que la longitud de onda a frecuencias más altas es más pequeña. Una vez resueltos los eventos de baja frecuencia, el contenido de alta frecuencia es gradualmente incluído en cada etapa para resolver los detalles de alta frecuencia. Con este procedimiento escalonado se puede compensar la necesidad de un

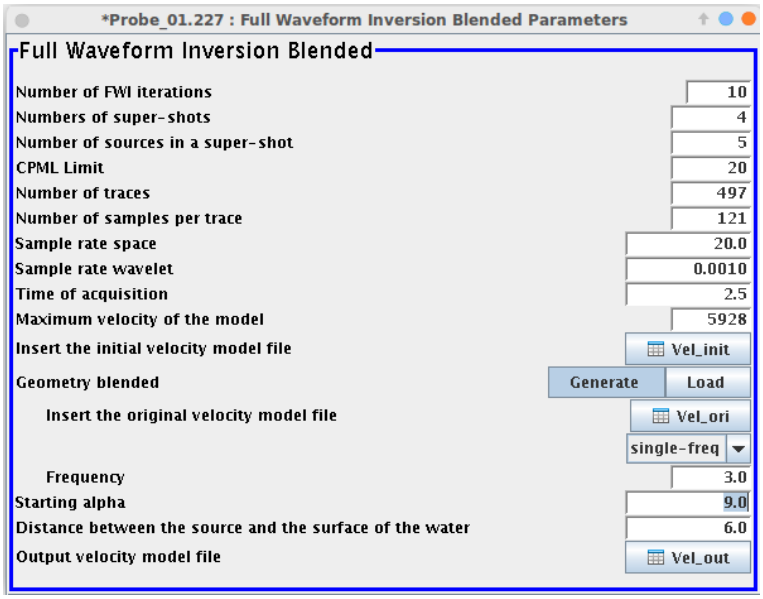

Figura 6.13 **Menú del módulo en ProMAX® seleccionando la opción *Generate* para una frecuencia.**

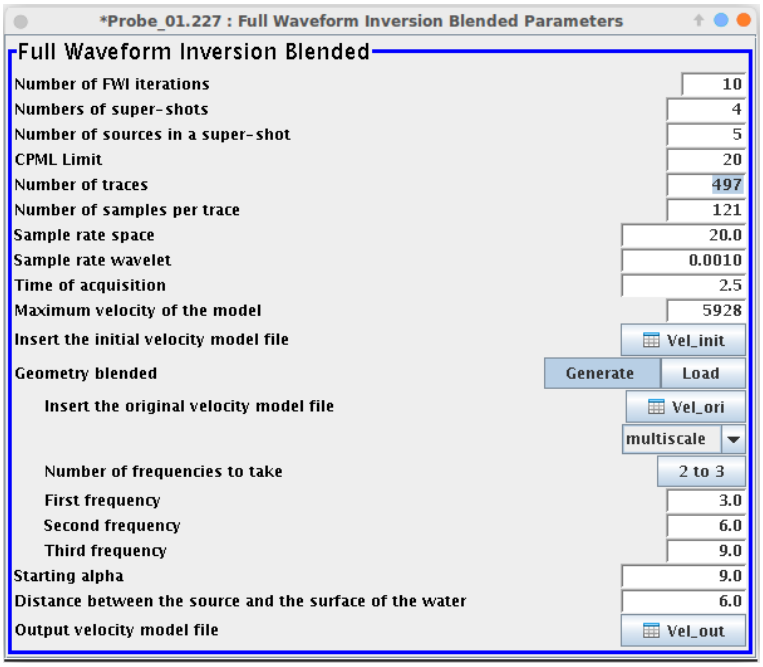

Figura 6.14 **Menú del módulo en ProMAX® seleccionando la opción *Generate* para multiescala de frecuencias.**

modelo inicial muy bueno, haciendo de la *FWI* una herramienta más robusta y factible en la industria sísmica (Dos Santos, A. & Pestana, R., 2015).

Según los resultados obtenidos por Flórez (2016) una geometría *blended* con 4 *super-shots* de cinco disparos, propagando las trazas observadas y modeladas aprovechando el principio de superposición es 1.8 veces más rápida que la implementación utilizando una

geometría tradicional. En este trabajo se propaga las trazas modeladas y observadas dentro de un mismo propagador para disminuir el tiempo de cómputo, en la tabla 6.3 se presenta una comparación del tiempo de propagación de un *super-shot* de cinco disparos y cinco disparos propagados individualmente, todos programados en un lenguaje CUDA-C. También se muestra el tiempo de cómputo de la *FWI* para 20 disparos (cuatro *super-shots*), usando ambas geometrías.

Tabla 6.3 Comparación del tiempo de cómputo requerido en una FWI con geometría blended y tradicional.

	Geometría *blended*		Geometría tradicional	
Frecuencia [Hz]	Tiempo de propagación [s]	Tiempo de *FWI* [s]	Tiempo de propagación [s]	Tiempo de *FWI* [s]
3	1.44	979.34	7.23	5003.37
6	1.42	992.95	7.32	5093.17
9	1.47	986.71	7.39	5111.53

Se puede notar que el tiempo de ejecución es menor para una adquisición en datos *blended* que para una adquisición tradicional, la propagación utilizando un solo propagador para cinco disparos es aproximadamente cinco veces más rápido que la propagación de cinco disparos de forma tradicional. Este resultado se debe a que el recorrido temporal en el modelo de velocidades se realiza una única vez para un grupo de disparos (contenidos en el *super-shot*) en el propagador propuesto, mientras que en una propagación tradicional este recorrido se realiza tantas veces como disparos se utilicen. El tiempo de la *FWI* usando geometría *blended* es 5.1 veces menor que en geometría tradicional, este valor se fundamenta en que las cinco propagaciones de las trazas observadas, modeladas y retropropagadas se realizan solo una vez dentro de la propagación del *super-shot*.

En la tabla 6.4 se presenta una comparación del modelo de salida obtenido con una geometría de adquisición sísmica *blended* (conformada por cuatro *super-shots* de cinco *shots*) y el hallado usando una geometría tradicional uniforme de 20 *shots* con el modelo original.

Tabla 6.4 Comparación del modelo obtenido con geometría blended y tradicional con el modelo original.

Frecuencia [Hz]	ECMN geometría *blended*	ECMN geometría tradicional
3	$36.9 * 10^{-3}$	$33.7 * 10^{-3}$
6	$34.2 * 10^{-3}$	$32.9 * 10^{-3}$
9	$33.7 * 10^{-3}$	$32.4 * 10^{-3}$

La geometría de adquisición sísmica *blended* presenta una ventaja en tiempo de ejecución en comparación con la geometría uniforme. Sin embargo, el grado de similitud entre el modelo de salida utilizando la geometría de adquisición *blended* y el modelo original se ve afectado por la interferencia entre disparos, por esta razón el ECMN entre el modelo original y el modelo obtenido con geometría *blended* es mayor que el hallado usando geometría tradicional uniforme aproximadamente en un 5.4%.

7. Trabajo futuro

Se propone realizar una codificación que seleccione la ubicación temporal y espacial de las fuentes dentro de cada *super-shot* teniendo en cuenta el porcentaje de *cycle skipping* como herramienta de validación que permita encontrar una distribución con una menor interferencia entre disparos.

8. Conclusiones

- Es posible implementar un módulo en ProMAX® que genere una geometría de adquisición *blended* usando posiciones y retardos en tiempo aleatorios, siguiendo un espaciamiento mínimo entre las fuentes de cada *super-shot*. El módulo diseñado en este trabajo también permite aplicar la *FWI* a una geometría de adquisición *blended* generada internamente o cargada de forma externa.
- Se compararon las implementaciones en MATLAB y ProMAX® presentando la evolución de la función de costo, el error cuadrático medio normalizado, un perfil 1D y un análisis de *Cycle Skipping*. Mostrando el modelo de velocidades obtenido, el cual se asemeja al resultado hallado en el código suministrado con un cuadrático medio normalizado de $8.6 * 10^{-3}$.
- En este trabajo de investigación se evaluó el tiempo de ejecución de las implementaciones en ProMAX® y MATLAB. La implementación en ProMAX® es 76.9 veces más rápida que la implementacion en MATLAB debido a que utiliza programación en paralelo.

9. Observaciones

- Para hacer un mejor uso de los recursos de la *GPU* en el propagador propuesto no se utilizan fronteras *CPML* de doble malla, debido a la cantidad de condicionales que

requiere esta forma de atenuación, los cuales pueden llegar a generar una divergencia de *threads* costosa computacionalmente.

- Se obtiene un tiempo de cómputo 5.1 veces menor en la *FWI* para 20 disparos (cuatro *super-shots* de cinco *shots*) con la geometría de adquisición *blended* implementada en comparación con 20 disparos con una geometría tradicional.
- Se corrobora que el grado de similitud entre el modelo obtenido con el modelo original es 5.4% menor en una medido con el ECMN entre el modelo original y para 20 disparos (cuatro *super-shots* de cinco *shots*) con la geometría de adquisición *blended* implementada en comparación con 20 disparos con una geometría tradicional.

Referencias bibliográficas

Abreo, D. (2015). *A practical implementation of acoustical Full Waveform Inversion on Graphical Processing Units*. Bucaramanga, Colombia.

Abreo, D. (2017). *Evaluación de estrategias de implementación de una inversión de onda completa (FWI) 3D acústica con densidad constante en el dominio tamporal sobre una arquitectura GPU*. Master's thesis. Bucaramanga, Colombia.

Abreo, D., Ramírez, A. & Abreo, S. (2017). A hybrid methodology for a 3D Full Waveform Inversion in time domain using GPUs. *15th International Congress of the Brazilian Geophysical Society EXPOGEf*. Brazil.

Alvarez, A. (2017). *Implementación de un control basado en CUDA*, Escuela Tecnica Superior e Ingeniería. Sevilla, España.

Bekhout, B. V. (2008). From simultaneous shooting to blended acquisition. *SEG Annual Meeting. Society of Exploration Geophysicists.*

Cornell University. (2017). *Virtual Workshop. Introduction to GPU and CUDA Programming*. Recuperado de https://cvw.cac.cornell.edu/gpu/thread_div?AspxAutoDetectCookieSupport=1

Dos Santos, A. & Pestana, R. (2015). Time domain multiscale Full-Waveform Inversion using the rapid expansion method and efficient step-length estimation. *Geophysics, vol 80,* R203-R216.

Flórez, K. (2016). *Inversión de Onda Completa (FWI) en tiempo para datos sísmicos adquiridos usando una geometría blended*. Bucaramanga, Colombia.

Jiménez, O. (2016). *Desarrollo de un módulo para ProMAX que realice el proceso de Inersión de Onda Completa (FWI) para la obtención de modelos de velocidades.* Bucaramanga, Colombia.

Kumar, R., Hannet W. & Felix, H. (2015). *Source separation for simultaneous towed-streamer marine acquisition - a compressed sensing approach*. Columbia, Canadá.

Monsegny, J. (2015). Conferencia: programación de módulos para ProMAX/SeisSpace. Bucaramanga, Colombia.

OGP. (2011). *An overview of marine seismic operations. 2011, vol. 448.* International Association of Geophysical Contractors.

Pasalic D. & Ray, M. (2010). Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations.

Perez, C. (2013). *Two-dimensional near - surface seismic imaging with surface waves: alternative methodology for waveform inversion*. PhD thesis, MINES ParisTech.

Plessix, R. (2006). A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, 395-503.

Serrano, J. (2017). *Evaluación de métricas en dominios transformados usadas en las metaheuristicas para generar un punto de partida favorable a la inversión de onda completa 2D*. Master's thesis. Bucaramanga, Colombia.

Serrano, J., Abreo S. & Ramírez A. (2017). A cycle-skipping analysis in transformed domains for Full Waveform Inversion using PSO. *15th International congress of the Brazilian Geophysical Society EXPOGEf*.

Souza *et al.* (2013). Full Waveform Inversion: Confronting Conventional and Blended Acquisition. . *Thirteenth International Congress of the Brazilian Geophysucal Society.* . Rio de Janeiro, Brazil.

Suarez, J. (2016). *Comparación de las herramientas de programación en paralelo OpenCL y CUDA para la implementación de la propagación de la ecuación de onda acústica 3D con densidad constante basada en STENCIL*. Bucaramanga, Colombia.

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

Yong, M. (2012). *Waveform-based velocity estimation from reflection seismic data*. Doctoral thesis, Colorado School of Mines.