Desarrollo de un módulo para ProMAX que realice el proceso de inversión de onda completa (FWI) para la obtención de modelos de velocidades.

OSCAR IVAN JIMENEZ BARANDICA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

Bucaramanga
2016

Desarrollo de un módulo para ProMAX que realice el proceso de inversión de onda completa (FWI) para la obtención de modelos de velocidades.

OSCAR IVAN JIMENEZ BARANDICA

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
Ph.D. Ana Beatriz Ramírez Silva
Co-director
Ph.D(c). Sergio Alberto Abreo Carrillo

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
Bucaramanga
2016

A Dios por bendecirme y permitirme
dar un gran paso en mi vida profesional.

A mis padres y seres queridos por el apoyo
incondicional a lo largo de este camino.

AGRADECIMIENTOS

Este trabajo es apoyado por la Compañía Colombiana de Petróleos ECOPETROL y COLCIENCIAS como parte del proyecto de investigación No. 0266-2013. El autor expresa su agradecimiento al grupo de investigación CPS de la Universidad Industrial de Santander. A mis directores Ana Beatriz Ramírez y Sergio Alberto Abreo por su presencia y apoyo incondicional. A los compañeros Julián Mantilla y Fabián Noriega por sus valiosos aportes que condujeron al desarrollo satisfactorio de este proyecto.

CONTENIDO

INTRODUCCIÓN 12
1. INVERSIÓN DE ONDA COMPLETA (FWI). 13
2. MÓDULO FWI EN PROMAX®. 15
2.1. PROMAX®. 15
2.2. DESARROLLO DE UN MÓDULO EN PROMAX®. 15
2.2.1. Creación de jerarquía de directorios 16
2.2.2. Programa 19
2.2.3. Makefile 22
2.2.4. Menú 24
2.3. INTEGRACIÓN DEL ALGORITMO FWI EN PROMAX®. 28
2.3.1. Interfaz gráfica 32
3. RESULTADOS 35
3.1. PRUEBAS 36
4. CONCLUSIONES 40
REFERENCIAS 42
BIBLIOGRAFÍA 44

LISTA DE FIGURAS

Figura 1. Problema inverso teórico. 13
Figura 2. Diagrama de proceso para desarrollar módulos en ProMAX®. 16
Figura 3. Jerarquía de directorios de ProMAX®. 18
Figura 4. Visualización en la interfaz del Código 4 en ProMAX®. 26
Figura 5. Lista de procesos del Código 5. 28
Figura 6. Enlace entre el módulo y el código de fwi. 29
Figura 7. Parámetros del módulo FWI en ProMAX®. 32
Figura 8. Estructura de la malla asociada al modelo inicial de velocidades. 33
Figura 9. Modelo de velocidades Marmousi (5.25Km x 1.7Km). 35
Figura 10. Modelo inicial de velocidades (5.25Km x 1.7Km). 35
Figura 11. Flujo de proceso de SeisSpace® ProMAX® para calcular la FWI. 36
Figura 12. Parámetros del módulo *Trace Display* de ProMAX®. 37
Figura 13. Modelo final de velocidades en ProMAX® para 100 fuentes y α=9. 38
Figura 14. Modelo final de velocidades sin ProMAX® para 100 fuentes y α=9. 38
Figura 15. Evolución función error ϕ(m) para 100 fuentes y α=9. 38
Figura 16. 1D profile para 100 disparos y α=9 con ProMAX® y ProMAX®. 39

LISTA DE TABLAS

Tabla 1. Variables globales comunes. 20

RESUMEN

Título:

DESARROLLO DE UN MÓDULO PARA PROMAX QUE REALICE EL PROCESO DE INVERSIÓN DE ONDA COMPLETA (FWI) PARA LA OBTENCIÓN DE MODELOS DE VELOCIDADES.[1]

Autores: Oscar Ivan Jimenez Barandica[2]

Palabras Claves: SeisSpace® ProMAX®, módulo SeisSpace® ProMAX®, Inversión de onda completa, Unidades de procesamiento gráfico.

Una etapa del procesamiento de datos sísmicos es la obtención de modelos de velocidades del subsuelo. Este proceso puede realizarse mediante métodos tales como la inversión de onda completa ó FWI (por sus siglas en inglés *full waveform inversión*). El algoritmo de la inversión de onda completa (FWI) se caracteriza por estimar modelos de velocidades de alta fidelidad que sirven como punto de partida para posteriores etapas, con el propósito de adquirir una imagen del subsuelo que se ajuste a la realidad. En la práctica estos procesos se llevan a cabo mediante software diseñados especialmente para aplicaciones geofísicas. Actualmente la mayoría de las empresas petroleras Colombianas utilizan SeisSpace® ProMAX®. Sin embargo, no existe un módulo en este software que estime modelos de velocidades de gran precisión como lo hace el método FWI. En este proyecto de investigación se integró el algoritmo de FWI desarrollado en lenguaje de programación CUDA-C en un módulo de ProMAX®, estableciendo un conjunto de reglas de compilación en un archivo Makefile. El módulo ejecuta el algoritmo FWI sobre una unidad de procesamiento gráfico GPU (por sus siglas en inglés *graphics processing unit*) que minimiza los tiempos de ejecución en la obtención de modelos de velocidades del subsuelo.

---

[1] Trabajo de Grado modalidad en investigación

[2] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: P.hD. Ana Beatriz Ramírez. Codirector: P.hD(c). Sergio Alberto Abreo.

ABSTRACT

TITLE:

DEVELOPMENT OF A MODULE FOR PROMAX TO PERFORM THE PROCESS TO FULL WAVEFORM INVERSION (FWI) FOR OBTAINING SPEED MODELS.[3]

AUTHOR: Oscar Ivan Jimenez Barandica[4]

KEYWORDS: SeisSpace® ProMAX®, module SeisSpace®, ProMAX® *full waveform inversion*, *graphics processing units*.

A stage of processing of seismic data is to obtain subsurfce models of speeds. This can be done by methods such as full waveform inversion (FWI). The algorithm full waveform inversión (FWI) is characterized by estimating models of speeds that serve as a starting point at a later stage, in order to acquire an image of the subsoil that fits reality. In practice, these processes are performed by softwares designed for geophysical applications.
Currently, most of the colombian oil companies use SeisSpace® ProMAX®. However, there is no a module in this software deemed accurate models of speeds as does the method FWI. In this research project was integrated the algorithm FWI, developed in CUDA-C language programming; on a module of ProMax®, establishing a set of compile rules into a file "Makefile". The module runs the algorithm FWI on a graphics processing unit (GPU) that minimizes the times of execution in obtaining subsurface velocity models.

---

[3]Degree Project
[4]Faculty of Physics Mechanics Engineering. Electrical, Electronics Engineering and Telecommunications School. Director: P.hD. Ana Beatriz Ramírez. Codirector: P.hD(c). Sergio Alberto Abreo.

INTRODUCCIÓN

La exploración de los hidrocarburos tiene como desafío encontrar imágenes del subsuelo, confiables y de alta resolución; para llevar a cabo actividades de perforación seguras y de bajo costo. Estas imágenes pueden ser adquiridas con el procesamiento de datos sísmicos. Una de las etapas del procesamiento es la obtención de modelos de velocidades que se pueden estimar a través de algoritmos computacionales.

Luego de validar el algoritmo de FWI a nivel experimental, usado para estimar modelos de velocidades de gran fidelidad, se requiere que este sea integrado a los flujos de trabajo usados por la industria. Actualmente SeisSpace® ProMAX®[5] [1], desarrollado por Landmark[6] [2]; es el software de procesamiento de datos sísmicos más usado por la industria de los hidrocarburos en Colombia.

En este trabajo se presenta el desarrollo de un módulo en ProMAX®, que integra el algoritmo de la FWI desarrollado en lenguaje CUDA-C en el entorno de procesamiento de SeisSpace®.

---

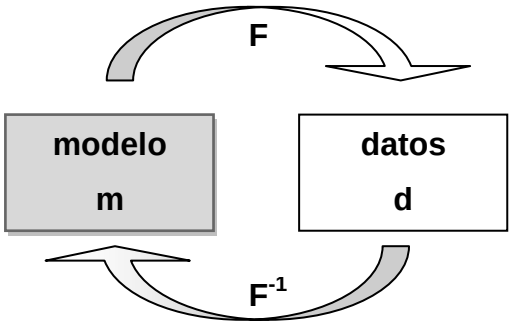
[5] A partir de esta sección se usará ProMAX® para hacer referencia a SeisSpace® ProMAX®

[6] *"Landmark, una línea de negocios de Halliburton,es el proveedor líder de tecnología de información y análisis, ciencia, software y servicios para la industria de exploración y producción"*.

1. INVERSIÓN DE ONDA COMPLETA (FWI).

La FWI es un método de inversión iterativo, que estima parámetros de un modelo del subsuelo **m**, para nuestro caso, velocidad de propagación del sonido a partir de los datos sísmicos adquiridos **d**. El objetivo es obtener el mínimo error cuadrático medio entre los datos adquiridos y los datos modelados generados a través de un operador lineal **F** (Ver Figura 1) [3].

Figura 1. Problema inverso teórico.



Fuente: Adaptado de [4].

La FWI requiere minimizar una función costo ó función de error $\phi(m)$, mediante la solución iterativa de dos problemas: 1) resolver el problema directo (de modelado de la solución de la ecuación de onda) y 2) el problema inverso (de actualización del modelo de velocidades). La función costo $\phi(m)$, representa el error entre los datos modelados y los datos observados, utilizando la norma ℓ_2 [5]:

$$\phi(m) = \frac{1}{2}\sum_{x=0}^{N_x}\left\|d(\boldsymbol{x}=x,\boldsymbol{z}=0,t|m) - d^{obs}(\boldsymbol{x}=x,\boldsymbol{z}=0,t)\right\|_2^2, \qquad (1)$$

donde $d^{obs}(x=x,z=0,t)$ representa los datos observados en una localización espacial dada $(x=x,z=0)$ y $d(x=x,z=0,t|m)$ representa los datos modelados en las mismas coordenadas espaciales, usando un modelo de velocidades conocido m, de dimensiones N_x por N_z. Los datos modelados se obtienen

mediante la solución numérica de la ecuación de onda (ecuación 2) usando diferencias finitas [5].

$$\frac{1}{m^2(x,z)}\frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial z^2}. \tag{2}$$

El modelo de velocidades se actualiza iterativamente usando [5]:

$$m^{k+1} = m^k - \propto \boldsymbol{I} G\left(m^k\right), \tag{3}$$

donde $k+1$ es la iteración siguiente, k es la iteración actual, α es una constante y $G(m^k)$ es el gradiente, el cual se puede aproximar como propusó Plessix [6]:

$$G\left(m^k\right) = -\sum_s \int_0^T q_s(x,z,T-t)\frac{\partial^2 p_s(x,z,t)}{\partial t^2}dt, \tag{4}$$

"Donde p_s es el conjunto de trazas obtenidas a partir de fuentes puntuales en la superficie y q_s es el conjunto de trazas obtenidas a partir de la ecuación de onda 2D acústica pero usando como fuente el error entre los datos modelados y observados, en el tiempo inverso. Por esta razón, q_s se llama el campo retro-propagado del residual" [4].

2. MÓDULO FWI EN PROMAX®.

2.1. PROMAX®.

ProMAX® es un software de procesamiento de datos sísmicos desarrollado por Landmark; su función principal es crear, modificar y ejecutar los flujos de procesamiento. Un flujo es una secuencia de procesos que se construyen mediante la selección de los procesos que contiene ProMAX® para manipular los datos sísmicos. Un flujo típico contiene un proceso de entrada, uno o más procesos de manipulación de datos y un proceso de salida [7].

2.2. DESARROLLO DE UN MÓDULO EN PROMAX®.

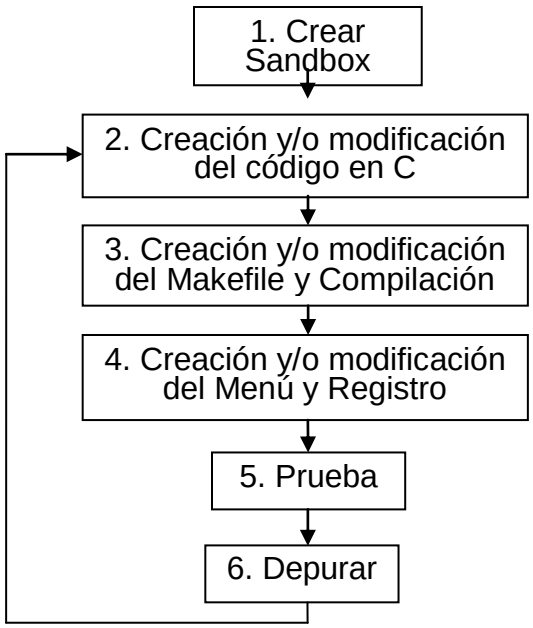
ProMAX® proporciona una plataforma para el desarrollo de software futuros que permite integrar algoritmos de procesamiento en el entorno de SeisSpace® para crear flujos de trabajos personalizados. Estos algoritmos son añadidos a ProMAX® a través de módulos desarrollados en el ambiente de programación Fortran ó C. Sin embargo es posible integrar algoritmos de procesamiento desarrollados en lenguajes de programación cuyo compilador puede funcionar con un comando de *Shell*[7] haciendo uso de un archivo Makefile.

Para el desarrollo de un módulo en ProMAX®, se debe seguir los pasos descrito en el diagrama de la Figura 2 [8].

---

[7] Shell: término usado para referirse al intérprete de comandos de sistemas operativos basados en Unix y similares como GNU/Linux.

Figura 2. Diagrama de proceso para desarrollar módulos en ProMAX®.



En el diagrama de la Figura 2, se muestran dos pasos fundamentales para el desarrollo de módulos en ProMAX®, el menú y el programa (código en C). El **menú** es la ventana que aparece en la interfaz de usuario para que el cliente introduzca los parámetros al programa desarrollado. El **programa** es el código que se ejecuta realmente y se obtiene de compilar el **código en C** [7].

2.2.1. Creación de jerarquía de directorios. Para ayudar a darle orden a la programación en ProMAX®, se recomienda crear una jerarquía de directorios de ProMAX® en el directorio personal, utilizando el programa Makeadvance[8].

Para crear la jerarquía de directorios es necesario definir un conjunto de variables de entorno. En sistemas operativos basados en Linux se usa el shell bash (~/.bashrc).

```
>>   sudo gedit ~/.bashrc
```

[8] Makeadvnce es un sell scrip que se encuentra en $PROMAX_HOME/puerto/bin que se usa para la creación de la jerarquía de directorios de ProMAX®.

Usando un editor de texto se verifica que la variable PROMAX_HOME[9] sea igual a la ruta del directorio donde está instalado ProMAX®, además se deben escribir los siguientes alias antes de ejecutar el comando Makeadvance:

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

una vez se tiene lo necesario en el (~/.bashrc), se carga en la consola con el comando:

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

Finalmente se ejecuta el script Makeadvance que produce un espejo de la jerarquía de directorios de $PROMAX_HOME en el directorio personal, pero sin copiar los archivos (Ver Figura 3). Esto permite trabajar en el desarrollo de módulos de ProMAX® bajo el directorio personal.

```
>>   Makeadvance
```

Una vez creada la jerarquía de directorios, se procede a incluir una nueva variable de entorno en el bashrc (~/.bashrc) con la ruta del directorio donde está instalado ProMAX®, particularmente, en este trabajo se usará la variable $MY_PROMAX_HOME como se muestra a continuación:

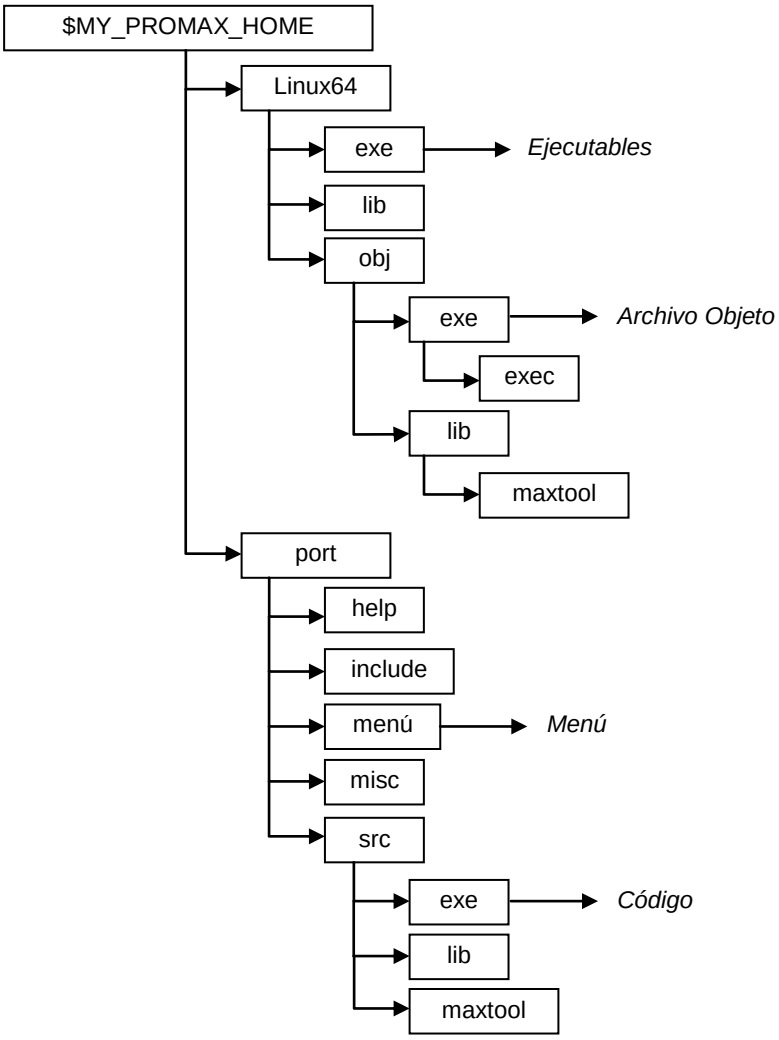
```
export MY_PROMAX_HOME=~/”ruta del sistema de ficheros”
```

a partir de este momento se usará MY_PROMAX_HOME para hacer referencia a la ubicación donde se desarrollarán los módulos de ProMAX®.

---

[9] $PROMAX_HOME es una variable de entorno y corresponde a la ruta del directorio donde está instalado el software ProMAX®.

Figura 3. Jerarquía de directorios de ProMAX®.



Para empezar a desarrollar un módulo, se debe crear un directorio en la ubicación MY_PROMAX_HOME/port/src/exe, preferiblemente con el nombre de la herramienta que se desarrollará, particularmente para este trabajo lo nombraremos fwi:

```
>>   cd $MY_PROMAX_HOME/port/src/exe
>>   mkdir fwi
>>   cd fwi
```

En el directorio previamente creado, se almacena el codigó de C, el Makefile y los códigos de los algoritmos desarrollados en lenguajes de programación que permiten ser integrados[10] a un módulo de ProMAX®.

2.2.2. Programa. El programa ejecuta una serie de instrucciones escritas en un lenguaje de programación, con la finalidad de resolver una o varias tareas. Actualmente, los módulos de ProMAX® son escritos en el entorno de programación C.

El programa es el producto de compilar un código, cuya extensión es **.c** y debe guardarse en el directorio (fwi) creado en la sección anterior [7].

En el Código 1 se presenta una estructura básica del programa de un módulo de ProMAX®.

Código 1. Estructura general del código .c.

```
1  //Header ProMAX
2  #include <cglobal.h>
3  #include <cpromax.h>
4  #include <cSocketTool.h>
5
6  //Main
7  int main(int ac, char **av)
8  {
9    initStandAlone(ac,av, "ST_NOOP");
10        stConnectToServer();
11        stSetToolName("No operation");
12
13        //Initialization code goes here
14
15        stEndInitialization ();
16
17        //Processing code goes here using stGetTrace,
18        //stGetEnsemble, stPutTrace and stPutEnsemble
19
```

[10] Los algoritmos que se pueden integrar a un módulo de ProMAX®, son los desarrollados en un lenguaje de programación cuyo compilador puede funcionar con un comando de *Shell.*

```
20        stCloseSocketLink();
21        XtDisplay();
22        return EXIT_SUCCESS;
23   }
```

Librerías de ProMAX®: ProMAX® contiene tres importantes archivos con extensión **.h** que corresponden a librerías exclusivas del software: **cglobal.h**, **cpromax.h** y **cSocketTool.h**. Estos archivos están ubicados en el directorio PROMAX_HOME/puerto/include.

El archivo **cglobal.h** permite acceder a las variables globales, que comúnmente son asignadas a los parámetros iníciales de los datos. En la Tabla 1 se muestran algunas de estas variables, las cuales son usadas en este trabajo.

Tabla 1. Variables globales comunes.

Variable Global	Tipo	Descripcion
samprat	Float	Tiempo de muestreo en milisegundos
numsmp	Int	Número de muestras por traza
maxdtr	Int	Número máximo de trazas
nth	Int	Número de cabeceros de las trazas
idomain	Int	Tipo de dominio

Algunas de estas variables tienen opciones definidas en el archivo **cglobal.h** que deben ser seleccionadas por el desarrollador de acuerdo al problema y lo establecido en el archivo **cglobal.h** (ver archivo **cglobal.h**). A continuación se muestra un ejemplo para asignar un valor a una variable global en un código:

```
globalRuntime->idomain=IDEPTH
```

donde *idomain* es la variable global, para este ejemplo corresponde al tipo de dominio y *IDEPTH* es la opción que ajusta el tipo dominio: espacio vs profundidad.

El archivo **cpromax.h** contiene la declaración de las librerías estándar de C, tales como **math.h**, **stdlib.h**, **stdio.h**, etc. También contiene definiciones de las

funciones de ProMAX®, para crear y/ó modificar tablas y bases de datos del software. Se puede obtener información de las rutinas ejecutando desde la terminal de Linux el comando *aman* como se muestra a continuación:

```
>>   aman "nombre de la función"
```

El archivo **cSocketTool.h** contiene funciones tales como *stConnectToServer* que permite establecer la conexión con SeisSpace® ProMAX® y *stCloseSocketLink* cierra la conexión con ProMAX®, entre otra rutinas que también se puede acceder a la descripción usando el comando *aman*.

Inicialización del código: en el apartado "*Initialization code goes here*" del Código 1, se declaran las variables que se usarán en el módulo, y se asignan a los parámetros de entrada que se establezcan en el archivo **.menu**. Una forma de realizar la declaración y asignación de estas variables se muestra a través del siguiente ejemplo:

```
Float dhh;
uParGetFloat("DHH", &dhh);
```

En donde *dhh* es la variable que se declara y *DHH* es el variable que almacena el parámetro ingresado por el usuario desde el menú, este parámetro es asignado a la variable *dhh* a través de la rutina *uParGetFloat*[11].

Procesamiento: en el Código 1, después de la función *stEndInitialization,* se realiza el procesamiento de los datos, en esta parte del código se pueden realizar cálculos numericos y hacer uso de rutinas de procesado de datos.

Finalmente, se cierra la conexión con ProMAX®, usando la rutina *stCloseSocketLink.*

El programa se obtiene al compilar el código en C, usando una archivo Makefile.

---

[11] Ejecutar en la terminal de comandos de Linux: >> aman uParGetFloat

2.2.3. Makefile. El Makefile es un archivo que especifica un conjunto de reglas para determinar que partes de un programa necesitan ser recompiladas y que comandos necesita para recompilarlas.

Se puede utilizar con cualquier lenguaje de programación cuyo compilador puede funcionar con un comando de *shell*.

El make utiliza su base de datos y los tiempos de la última modificación de los archivos para decidir cuál de los archivos necesitan ser actualizados (recompilados) y cómo. En un programa, el archivo ejecutable se actualiza desde archivos objetos, que a su vez son generados por compilar un archivo de código fuente que puede usar librerías de sistema [9].

La compilación se refiere al proceso de convertir un programa desde el código fuente en código de máquina, una serie de datos digitales que son utilizados para controlar la Unidad Central de Procesos (CPU) del ordenador. Este código de máquina es almacenado en un fichero conocido como fichero ejecutable [10].

El archivo Makefile que se usa para compilar los códigos de ProMAX®, se obtiene de los ejemplos desarrollados por ProMAX® ubicados en los directorios MY_PROMAX_HOME/port/src/exe/stand_alone1 ó MY_PROMAX_HOME/port/src/exe/stand_alone2 (ver Código 2). El archivo Makefile se debe guardar en la misma ubicación del Código en C.

Código 2. Estructura general del código Makefile

```
1  include advance.make
2
3  # Variable define
4  srcs =  $(newsrcs)
5  objdir = objexedir
6  srcdir = srcexedir
7  parentdir =
8  uobjs = $(define_uobjs)
9  usrcs = $(define_usrcs)
10   aobjs = $(define_aobjs)
11   exe = $(exedir)/$(parentdir).exe
12   libs = $(AGCLITELIBS)
```

```
13
14   # Rules
15   all :
16        $(UPDATE_LIBRARIES)
17        $(MAKE) $(exe)
18
19   $(exe) :$(uobjs) $(libs) $(LIBSCUDA)
20        $(MAKETARGETDIR);
21        $(C_LINK)  $(XMLIBS)  $(CLIBS)  $(SYSLIBS)  ||
   $(BACKUP_ON_ERROR)
22   $(objexedir)/$(parentdir)/%.o : %.c
23        $(ANSIC_COMPILE)
24   # Generic rules to clean up.
25   cleanexe:
26        @echo "Removing: $(exe)"; \
27        /bin/rm -f $(exe) 2> /dev/null
28   cleanobjs:
29        @echo "Removing: $(uobjs)"; \
30        /bin/rm -f $(uobjs) 2> /dev/null
31   clean: cleanexe cleanobjs
32        @echo "Removing: $($(objdir))/$(parentdir)"; \
33        /bin/rm   -rf   $($(objdir))/$(parentdir)   2>
   /dev/null
```

Para compilar el código de C, se deben modificar las lineas 4 y 7 del Makefile (Código 2) de la siguiente forma:

```
4    srcs = fwi.c $(newsrcs)
7    parentdir = fwi
```

donde ***fwi.c*** es el nombre del código desarrollado en C y ***fwi*** es el nombre del directorio donde se guarda el archivo ***fwi.c***.

Una vez realizadas las modificaciones correspondientes al archivo Makefile, se ejecuta el comando promake en la ubicación donde está guardado el Makefile desde la terminal de comandos de Linux para compilar el código en C.

```
>>   cd $MY_PROMAX_HOME/port/src/exe/fwi
>>   promake
```

Si al compilar se presentan errores, se debe revisar nuevamente el código en C para corregir los errores, pero sí no hay errores, es el momento de empezar a desarrollar el menú que se presentará en la interfaz de usuario de ProMAX®.

2.2.4. Menú. El menú es un archivo ASCII que contiene la descripción de los parámetros y reglas que controlan lo que presenta la interfaz de usuario, actualmente son escritos en *Lisp* y tiene extensión **.menu**, dichos archivos son interpretados en tiempo de ejecución, es decir; no necesitan ser compilados [7].
A diferencia del código fuente de los módulos incluidos en los productos de ProMAX®, se tiene acceso a todos los códigos de los menús de los módulos de ProMAX®, así que es posible estudiar y copiar esos archivos libremente, ubicados en el directorio $PROMAX_HOME/port/menu/promax. Es probable que en algún lugar dentro de los menús existentes, haya un menú que hace algo similar a lo que deseas hacer [7].

El archivo menú está conformado por 4 partes: *Menu heading, Parameter Specifications, Exec_Data y Rules* y se debe guardar en la ubicación MY_PROMAX_HOME/port/menu/promax. En el siguiente código se muestra una estructura general del código de un menú:

Código 3. Estructura general del código .menu.

```
1   '(
2     ;*******Menu heading********
3     name: PROGRAM_NAME
4     label: "Sample ProMAX Menú"
5     value_tab: 48
6     ;**Parameter Specifications**
7     parameter: PARM
8     ; Atributos de los parámetros
9     ;*********Exec_Data**********
10        exec_data: ("PROGRAM_NAME"
11                        ; Parámetros de ejecución
12                        )
13        ;***********Rules************
14        rules: (
```

```
15                    ; Reglas
16                   )
17    )
```

Título del menú: En el apartado *menu heading*, ***name*** hace referencia al nombre del programa al que pertenece el menú, esta linea es opcional, ***label*** corresponde al nombre que aparecerá en la lista de procesos ProMAX® en la interfaz de usuario, La cadena de caracteres del *label* debe coincidir con la del archivo de procesos (el archivo que contiene la lista de menús), o de lo contrario se emitirá un error en la ventana de ProMAX®.

Finalmente, ***value_tab*** es valor que especifica cuántos espacios de caracteres desde el extremo derecho del menú se ubicaran los cuadros de entrada de texto que permiten introducir o variar parámetros.

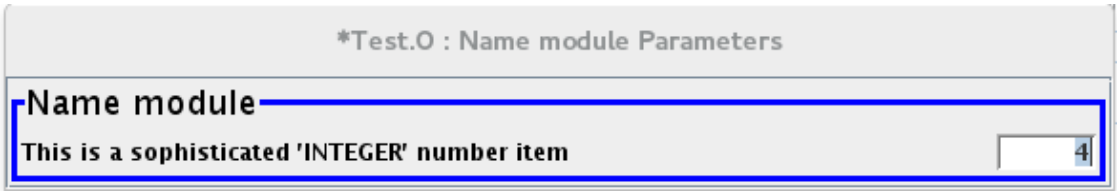
Especificaciones de los parámetros: Esta sección se utiliza para especificar los nombres, tipos y otros atributos de los parámetros del menú. A continuación se presenta un ejemplo sencillo de la escritura de un parámetro y algunos de sus atributos más relevantes.

Código 4. Ejemplo para la declaración de un parámetro del menú.

```
1 parameter: FANCY_INTEGER
2 text: "This is a sophisticated 'INTEGER' number
  item"
3 type: typein:
4 type_desc: ( int: 3 nil 999999 )
5 value: 4
6 mouse_text: "Mouse pointing help goes here (this
  integer is 8 characters                wide,
  test limits are 999999 (upper) with no lower
  limit)."
```

En la Figura 4 se muestra visualmente como se presenta en la interfaz de ProMAX®, el ejemplo anterior.

Figura 4. Visualización en la interfaz del Código 4 en ProMAX®.



En el Código 4, ***parameter*** corresponde al nombre de la variable en el menú, ***text*** es una cadena de caracteres que aparecerá para cada parámetro en la interfaz del menú de ProMAX®, ***type*** describe la clase de objeto que aparecerá en el menú. Sí el valor de ***type*** es ***typein*** (*type: typein:*), indica que el cuadro de entrada de datos aceptará la entrada a través teclado numérico, y buscará mas especificaciones en el atributo ***type_desc****.* El valor predeterminado para el parámetro se establece mediante el atributo ***value***.

Finalmente el atributo ***mouse_text****,* especifica el texto que aparece en un cuadro en la parte inferior del cursor cuando el puntero del mouse se ubica sobre un parámetro.

Exec_data: La primera linea en el *exec_data* es

```
exec_data: ("PROGRAM_NAME"
```

El paréntesis de apertura es el inicio de la agrupación del *exec_data,* hay un paréntesis de cierre al final de la sección. La cadena ***PROGRAM_NAME*** dentro de las comillas se debe reemplazar por el nombre del programa.

Los parámetros de ejecución indican que archivo **.exe** se ejecutará y que parámetros se presentarán al usuario.

A continuación se escribe un ejemplo de cómo se escribe un parámetro de ejecución de un programa y una variable:

```
("program" implicit: “Cprogram.exe”)
("FANCY_INTEGER" implicit: (value 'FANCY_INT))
```

Donde, Cprogram.exe hace referencia al programa que se va a ejecutar y FANCY_INT es la variable que almacena el dato introducido por el usuario, a través de esta variable es posible transferir la información la información al Código en C.

Reglas: La sección de reglas es la parte del archivo del menú que controla, entre otras cosas, qué parámetros están ocultos al usuario cuando no son necesarios. La sección de reglas comienza con la cadena "*rules (*" y termina con un paréntesis de cierre. Las reglas son de la forma

```
( nombre_regla ( condicional )
     (acción verdadera)
     (acción falsa)
)
```

La cadena ***nombre_regla*** representa la etiqueta de la regla, no puede contener espacios entre caracteres y debe tener al menos un carácter.

El ***condicional*** generalmente realiza la evaluación de un parámetro, en Lisp se realiza con la siguiente estructura

```
("Tipo de evaluación" ("Variable a evaluar") "Valor de
comparación")
```

Los ***Tipos de evaluación*** son >, <, =, etc. La ***variable a evaluar*** es algún atributo de un parámetro y el ***valor de comparación*** puede ser una constante o una expresión, como la suma de dos parámetros.
Las reglas pueden llegar a ser algo complejas en la medida en que se usen declaraciones compuestas como ***if*** o ***and****,* así como también funciones compuestas.

Instalar el menú: Una vez que se ha escrito el menú, se debe instalar en la lista de menús de ProMAX® para que aparezca en la lista de procesos de la interfaz de usuario. Se debe crear un archivo de texto con el nombre ***ProcessesPM***, en la

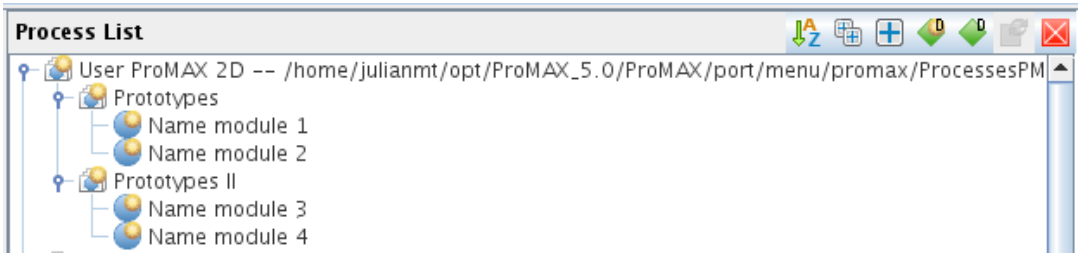
misma ubicación donde se guardo el archivo **.menu**. El ***ProcessesPM*** debe tener la siguiente estructura:

Código 5. Estructura del archivo ProcessesPM.

```
1  '(
2    ("Prototypes"
3         ("Name module 1" "NameMenu" h)
4         ("Name module 2" "NameMenu" h)
5    )
6    ("Prototypes II"
7         ("Name module 3" "NameMenu" h)
8         ("Name module 4" "NameMenu" h)
9  )
```

En el Código 5, ***Name module 1, Name module 2, Name module 3 y Name module 4***, Corresponden al nombre del módulo que se mostrará en la lista de procesos y ***NameMenu*** corresponde al mismo nombre asociado al código **.menu**. En la Figura 5 se muestra la como queda estructurada la lista de procesos del Código 5.

Figura 5. Lista de procesos del Código 5.



2.3. INTEGRACIÓN DEL ALGORITMO FWI EN PROMAX®.

El algoritmo FWI se incorpora al módulo de ProMAX® como una función externa, que se declara en el Código en C (Código 1) antes del ***main***, en términos generales la definición de una función se puede escribir como [11]:

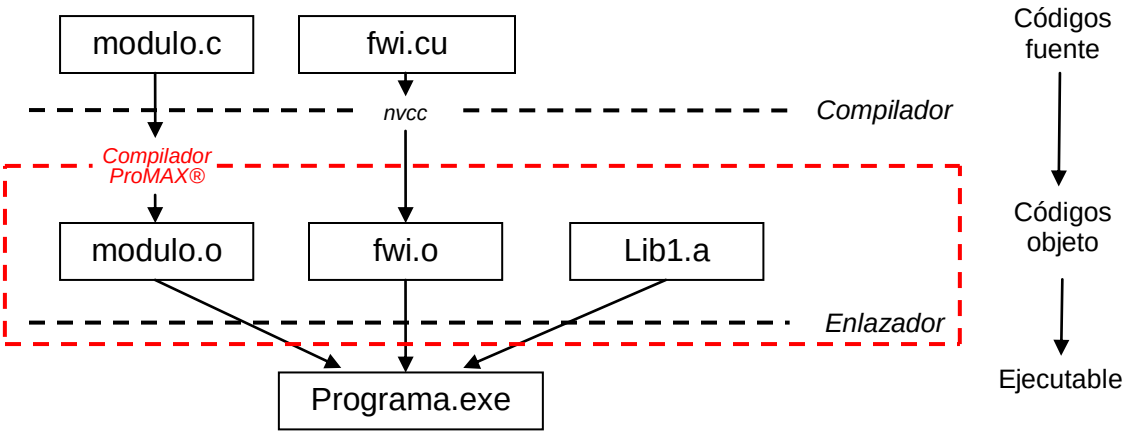
```
tipo_almacenamiento tipo_datos nombre(tipo1 arg1, tipo2
arg2,…,tipon argn)
```

donde ***tipo_almacenamiento*** denota el tipo de almacenamiento asociado con la función, ***tipo_datos*** el tipo de datos del valor devuelto por la función, ***nombre*** denota el nombre de la función, ***tipo1, tipo2, o, tipon*** se refieren a los tipos de los argumentos formales y ***arg1, arg2,.., argn*** a los argumentos formales.

Los parámetros de entrada del algoritmo de FWI, son cargados por el usuario desde ProMAX® y accedidos por el código FWI a través de punteros declarados como argumentos de la función.

Para compilar el código de FWI y el código del módulo de ProMAX®, se modificó el archivo Makefile proporcionado por el software. En el Makefile se incluyeron las instrucciones para compilar el código de FWI desarrollado en lenguaje de programación CUDA-C, cuya extensión es **.cu**. El compilador de CUDA-C es ***nvcc*** (ver Código 6).

Figura 6. Enlace entre el módulo y el código de fwi.



Los programas de un proyecto están compuestos de varios archivos de código fuente cuyas extensiones pueden ser diferentes. El resultado de compilar cada uno de estos códigos, es un archivo objeto con extensión **.o**, cuyo contenido es la traducción del código fuente a código de máquina. Finalmente un enlazador une

todos los archivos objeto para generar un ejecutable con todas las instrucciones programadas, cuya extensión es **.exe** (ver Figura 6) [12].
Escribir un Makefile es simplificado por las definiciones estándar y las normas contenidas en advance.make y otros archivos almacenados en el directorio make de ProMAX®. Advance.make establece numerosas variables para rutas de directorios que utiliza el Makefile, entre otras cosas. En particular, advance.make determina automáticamente el tipo de máquina y las variables de biblioteca, compiladores, banderas, etc. Un programador de ProMAX® no necesita escribir un Makefile desde cero, porque en el directorio de desarrollo local se le proporciona una plantilla que incluye las variables de los directorios, por lo tanto, se editará este archivo para integrar el algoritmo FWI desarrollado en CUDA-C.

La estructura del archivo make está divida en 2 partes fundamentales, definición de variables y reglas. La definición de una variable, especifica un valor textual de la misma, que habitualmente es la ruta de un directorio o archivo. Por otro lado, una regla dice cuándo y cómo reconstruir uno o más archivos (objetivos de la regla) (normalmente uno por cada regla). Separado por el signo dos puntos (*:*) del nombre de la regla, se listan las dependencias de los objetivos, es decir, los archivos de los que depende cada objetivo para poder ser construído, y finalmente en una línea aparte después de un carácter *TAB* (tabulador) se dan las órdenes mediante las cuales se reconstruirá el objetivo, estas órdenes son las mismas que interpreta el *shell* de *UNIX* [10].

En el Makefile mostrado a continuación y proporcionado por ProMAX®, se encuentran resaltadas en negrilla, las variables y reglas que deben incluirse para incorporar el código de la FWI. Las variables `srcs` y `parentdir` asocian el nombre del código fuente y nombre del directorio respectivamente, donde se ubica el código del módulo de ProMAX®. La variable `LIBSCUDA` indica la ubicación de la librería que se requiere para compilar el código FWI. La regla `$(exe)` tiene como objetivo construir un ejecutable, el cual depende de los archivos objeto que son el

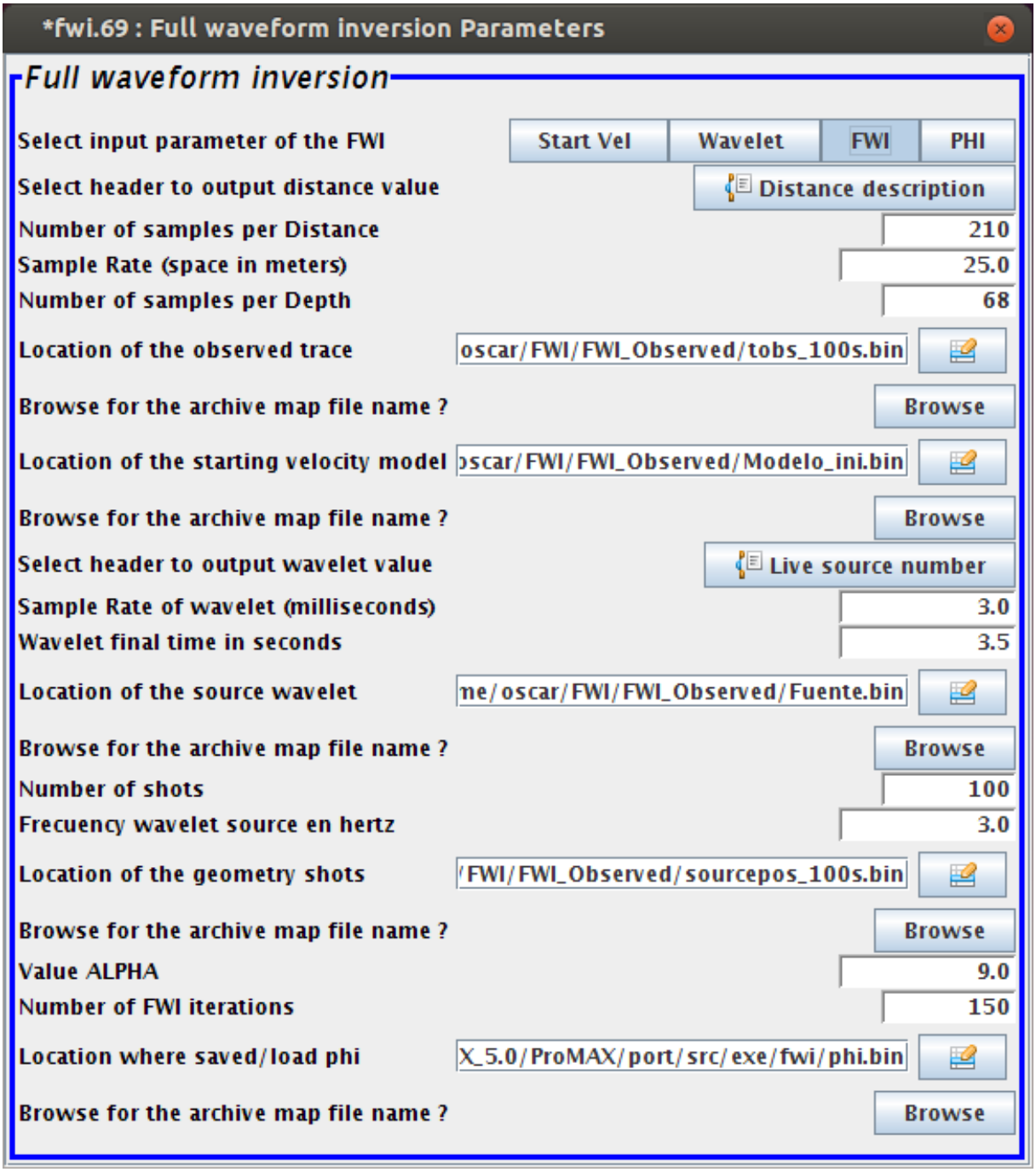
módulo creado y el código FWI (construido a partir de la regla **`FWI_cpml8_v1.o: FWI_cpml8_v1.cu`**.)

Código 6. Makefile para enlazar .c y .cu.

```
1  include advance.make
2
3  # Variable define
4  srcs = fwi.c $(newsrcs)
5  objdir = objexedir
6  srcdir = srcexedir
7  parentdir = fwi
8  uobjs = $(define_uobjs)
9  usrcs = $(define_usrcs)
10   aobjs = $(define_aobjs)
11   exe = $(exedir)/$(parentdir).exe
12   LIBSCUDA += /usr/local/cuda-7.0/lib64/libcudart.so
13   libs = $(AGCLITELIBS)
14
15   # Rules
16   all :
17        $(UPDATE_LIBRARIES)
18        $(MAKE) $(exe)
19
20   $(exe) :$(uobjs) FWI_cpml8_v1.o $(libs) $(LIBSCUDA)
21        $(MAKETARGETDIR);
22        $(C_LINK)  $(XMLIBS)  $(CLIBS)  $(SYSLIBS)  ||
   $(BACKUP_ON_ERROR)
23   $(objexedir)/$(parentdir)/%.o : %.c
24        $(ANSIC_COMPILE)
25   FWI_cpml8_v1.o: FWI_cpml8_v1.cu
26        nvcc FWI_cpml8_v1.cu -c
27   # Generic rules to clean up.
28   cleanexe:
29        @echo "Removing: $(exe)"; \
30        /bin/rm -f $(exe) 2> /dev/null
31   cleanobjs:
32        @echo "Removing: $(uobjs)"; \
33        /bin/rm -f $(uobjs) 2> /dev/null
34   clean: cleanexe cleanobjs
35        @echo "Removing: $($(objdir))/$(parentdir)"; \
36        /bin/rm   -rf   $($(objdir))/$(parentdir)   2>
   /dev/null
```

2.3.1. Interfaz gráfica. El menú que se desarrolló para el módulo de la FWI, se presenta en la interfaz de usuario de ProMAX® como se muestra en la figura 6.

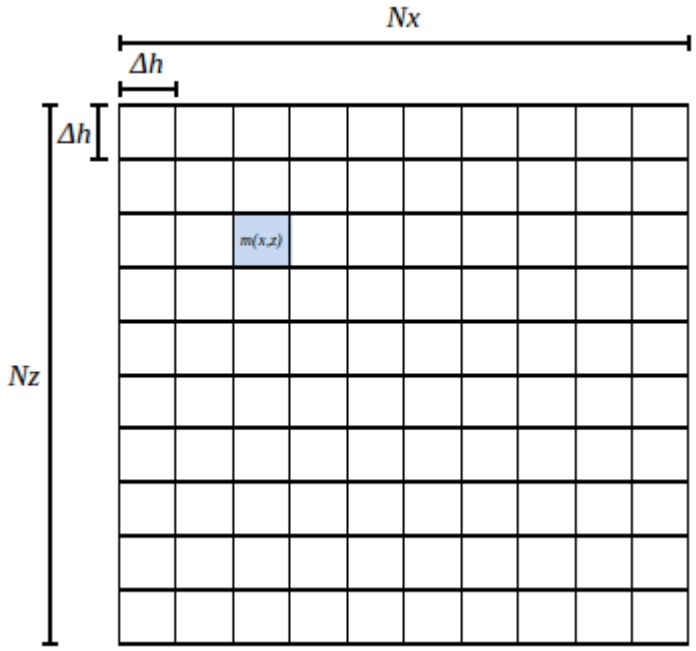
Figura 7. Parámetros del módulo FWI en ProMAX®.



En el módulo FWI: las trazas observadas, modelo inicial de velocidades, ondícula, geometría de los disparos y la función de costo $\phi(m)$; son archivos binarios que se cargan a ProMAX® a través de los botones "*Browse*" que permiten navegar a través del sistema de ficheros del ordenador y seleccionar el archivo binario correspondiente a los parámetros que requiere el módulo.

El módulo también requiere información adicional de cada uno de los parámetros de entrada FWI. El número de muestras por distancia, número de muestras por profundidad y frecuencia de muestreo espacial; son parámetros que se ingresan por el usuario para la correcta ejecución del flujo y están asociados a los modelos de velocidades inicial y final, muestreados equiespacialmente por un valor Δh en metros, de dimensiones N_x por N_z, donde N_x es el número de muestras por distancia, N_z es el número de muestras por profundidad y cada muestra es la velocidad correspondiente a la coordenada espacial (Ver Figura 8).

Figura 8. Estructura de la malla asociada al modelo inicial de velocidades.



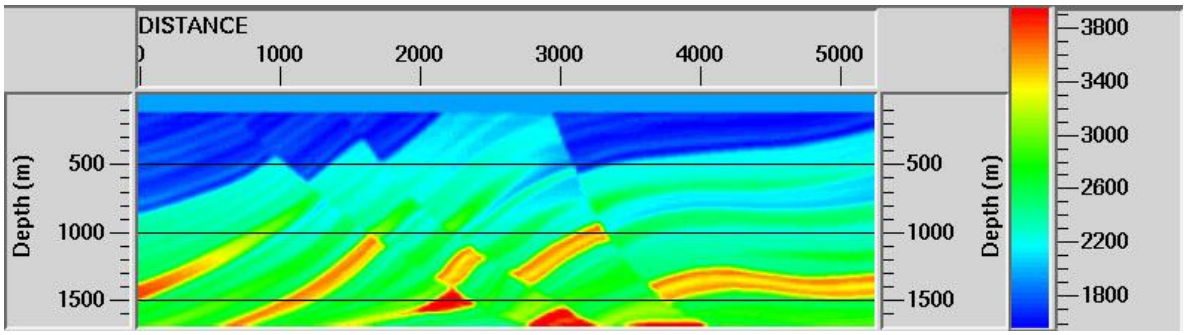
Los datos adicionales asociados a la "*wavelet*" son la frecuencia, el tiempo de muestreo y el tiempo de duración. Otro dato muy importante es el número de disparos y la ubicación donde estos se efectúan, esta información se carga a través de un archivo binario cuyo arreglo tiene la siguiente estructura $[x_1, z_1, x_2, z_2, \dots, x_n, z_n]$, donde x indica la distancia, z la profundidad y n corresponde a la cantidad de disparos.

Alpha ($\propto$) es una constante que también debe ingresarse por el usuario.
Los valores de phi $\phi(m)$ son almacenados en la ubicación y con el nombre que el usuario decida, cuyo nombre debe tener extensión **.bin**. Este archivo binario es creado una vez se ejecute el módulo FWI.

3. RESULTADOS

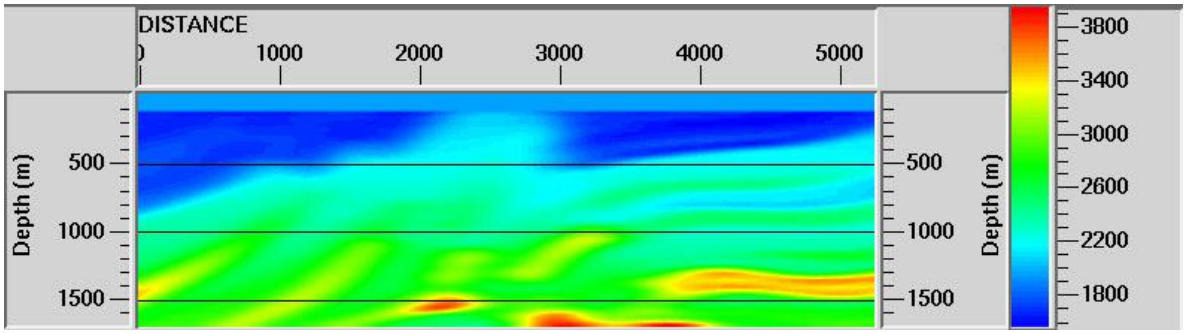
En 1988, el Institut Francais du Pétrole (IFP), creó el modelo de velocidad Marmousi **Fuente especificada no válida.**, del que se tomará una sección de 5.25km x 1.7km, como se muestra en la Figura 9. Para realizar las pruebas se ubicaron deliberadamente cien fuentes entre 250m y 5000m, igualmente espaciadas a una profundidad de 75m. Se generaron un conjunto de trazas observadas. Los geófonos están localizados en la superficie cada 25 metros, desde la posición 250m hasta la posición 5000m. Cada geófono registra 3.5 segundos en un intervalo de 3 milisegundos.

Figura 9. Modelo de velocidades Marmousi (5.25Km x 1.7Km).



El modelo inicial de velocidades para realizar las pruebas es el modelo Marmousi, luego de un suavizado horizontal de 5 puntos de 20 iteraciones y dimensiones 68x210, el modelo resultante se muestra en la Figura 10.

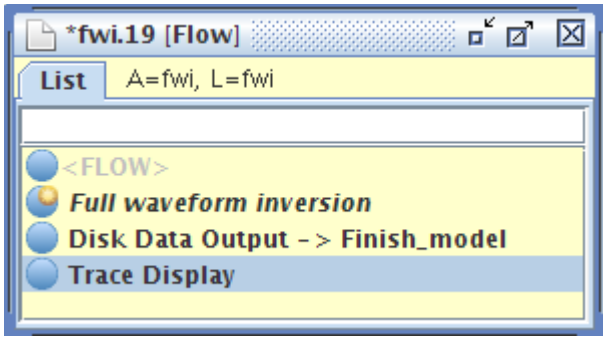
Figura 10. Modelo inicial de velocidades (5.25Km x 1.7Km).



3.1. PRUEBAS

Para realizar las pruebas se ejecuta el software ProMAX®. Se creó el flujo que se muestra en la Figura 10, y los parámetros que se usaron se muestran en la Figura 7.

Figura 11. Flujo de proceso de SeisSpace® ProMAX® para calcular la FWI.



La herramienta ***Trace Display*** que proporciona ProMAX®, se utiliza para graficar los resultados que se obtienen de los flujos de procesamiento. Particularmente, en este trabajo se usa para visualizar el modelos de velocidades que se obtuvo con el módulo FWI de ProMAX®. Los parámetros usados para el módulo ***Trace Display*** se muestran en la Figura 12.

Con el fin de realizar pruebas al módulo se seleccionaron deliberadamente los parámetros $\propto = 9$, $100\ shots$ y $150\ iteraciones$.

Con el propósito de validar el correcto funcionamiento del módulo de la FWI en ProMAX®, se llevo a cabo la misma prueba sin el módulo, es decir, se usaron los mismos parámetros de entrada para contrastar la evolución de la función de error phi $\phi(m)$ obtenida con ProMAX® y sin ProMAX®. Finalmente se compararon perfiles de velocidades transversales en una dimensión a 3 Km, para corroborar la similitud de los modelos obtenidos con ProMAX® y sin ProMAX®.

Figura 12. Parámetros del módulo *Trace Display* de ProMAX®.

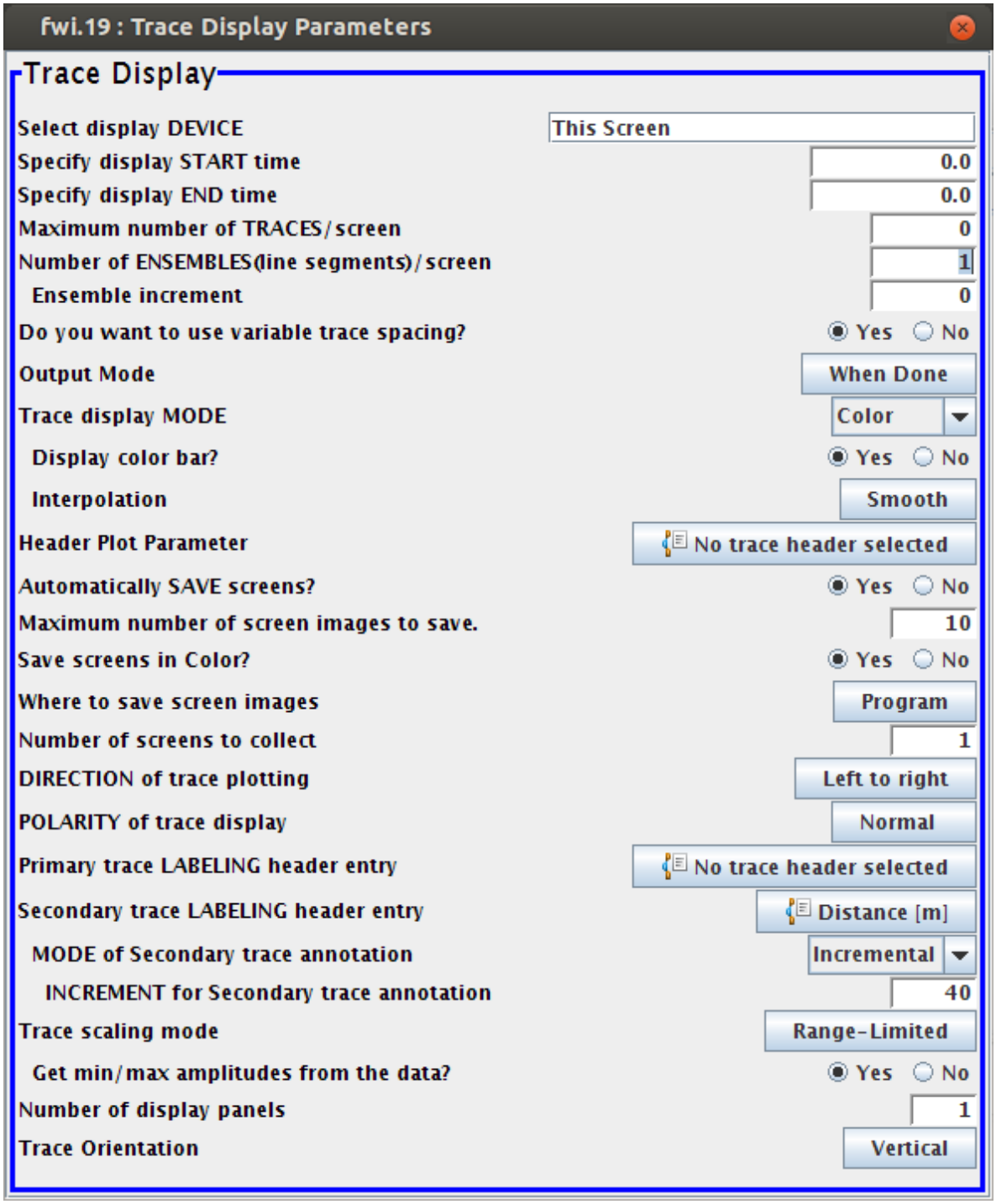


Figura 13. Modelo final de velocidades en ProMAX® para 100 fuentes y α=9.

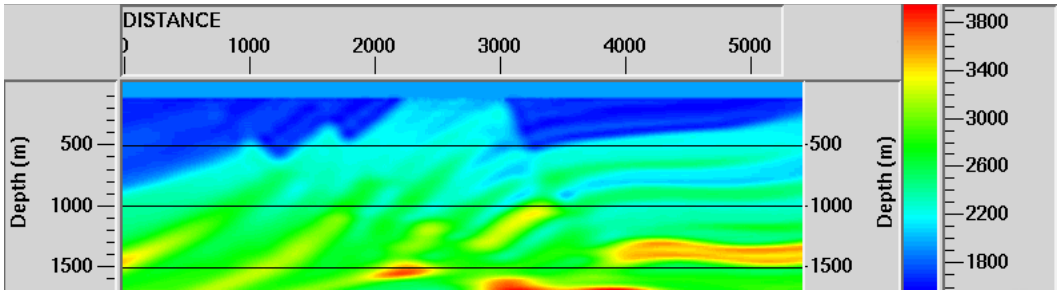


Figura 14. Modelo final de velocidades sin ProMAX® para 100 fuentes y α=9.

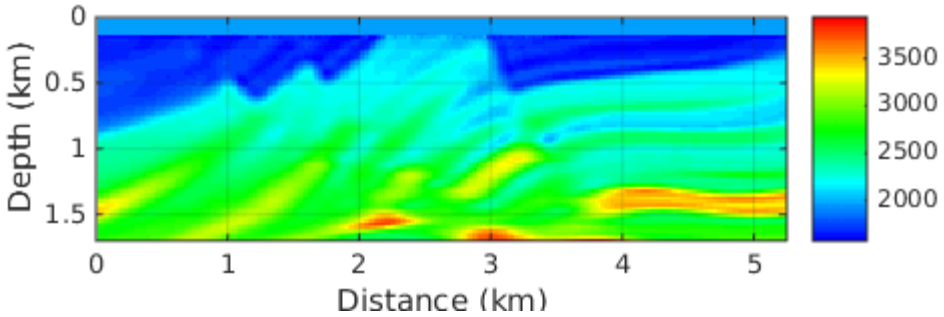
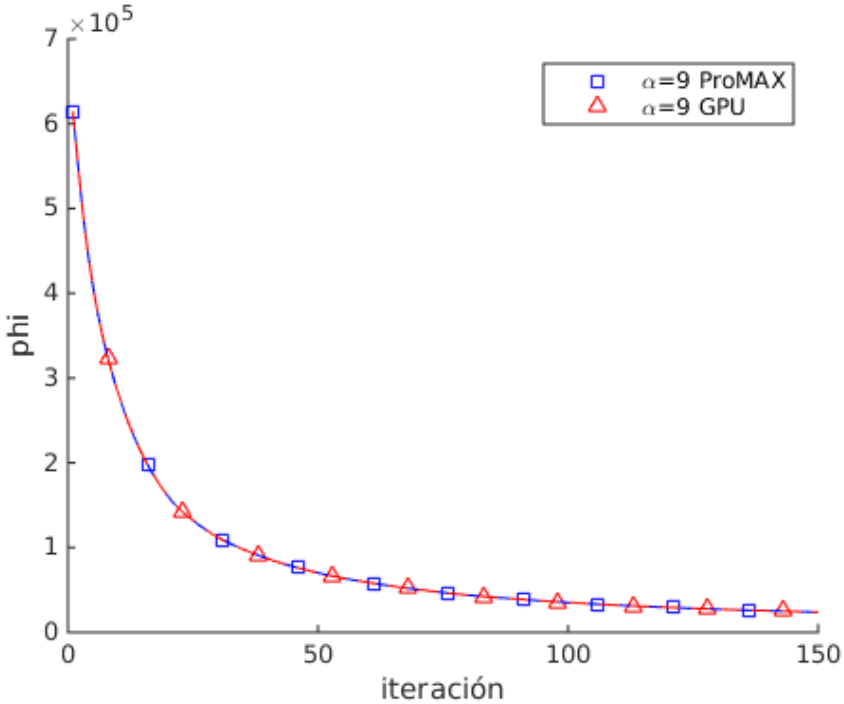
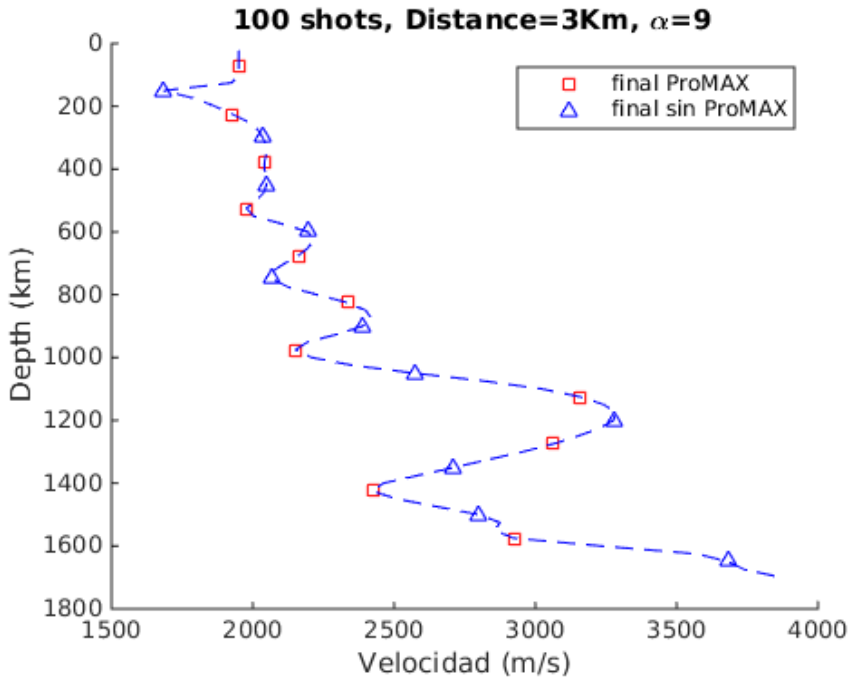


Figura 15. Evolución función error ϕ(m) para 100 fuentes y α=9.



Observando la evolución de las funciones de error $\phi(m)$ obtenidas al ejecutar el módulo de FWI y el algoritmo de FWI en la terminal de comandos de Linux, se corroboró la correcta integración del algoritmo de FWI en el módulo de ProMAX® (ver Figura 15)

Figura 16. 1D profile para 100 disparos y α=9 con ProMAX® y ProMAX®.



También se puede afirmar que los dos modelos de velocidades obtenidos en las pruebas son los mismos, realizando un análisis del perfil de velocidades en una dimensión (*1D profile*) a 3 Kilómetros de distancia ya que es donde se presentan las mayores variaciones de las velocidades con respecto a la profundidad. En la Figura 16 se observa que el perfil en una dimensión obtenido con el módulo de FWI en ProMAX® es igual al obtenido ejecutando el código de FWI en la terminal de comandos de Linux.

4. CONCLUSIONES

1. El kit de desarrollo de software de ProMAX® no solo permite integrar algoritmos desarrollados en FORTRAN o C, sino que también permite integrar cualquier código, cuyo lenguaje de programación puede ser compilado con un comando *shell*. La integración de los algoritmos se lleva a cabo a través de una función externa de C, ya que gran parte del código de nivel de ProMAX® está escrito en lenguaje FORTRAN y C; finalmente las instrucciones de compilación se incorporan en un archivo **make** (*Makefile*), el cual es proporcionado por ProMAX®. Particularmente en este trabajo se integra un algoritmo que se ejecuta sobre una unidad de procesamiento grafico y desarrollado en lenguaje de programación *CUDA-C*.

2. Se validó la similitud de la evolución de la función de costo $\phi(m)$ obtenida con el algoritmo FWI implementado en el módulo de ProMAX® y la obtenida con el código fuente ejecutado desde la terminal de LINUX, así mismo, se comprobó la igualdad entre perfiles transversales de velocidad de los modelos calculados con ProMAX® y sin ProMAX®, razón por la cual se corrobora la efectiva integración del algoritmo FWI en ProMAX® y se garantiza el correcto funcionamiento del módulo desarrollado, ya que se puede presentar un mal manejo de las estructuras que implica transferir los datos al algoritmo de FWI de una forma incorrecta, ocasionando una estimación errónea del modelo final de velocidades.

3. La universidad Industrial de Santander, particularmente el grupo de Investigación CPS[12], son pioneros desarrollando módulos en ProMAX® que

[12] Grupo de investigación CPS, Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones, Universidad Industrial de Santander, Bucaramanga, Colombia

integren algoritmos realizados en lenguaje de programación CUDA-C, que permiten agilizar los tiempos de respuesta en el procesamiento de los datos sísmicos.

4. En este trabajo, las trazas sísmicas (*Trazas observadas*) se cargan directamente desde el módulo, se propone como trabajo futuro asociar el módulo de la FWI con el módulo *Seg-Y* que tiene ProMAX®, que permiten cargar las trazas sísmicas y los *header* de las trazas y posteriormente transferir los datos al proceso FWI.

REFERENCIAS

[1] Halliburton|Landmark. (2015, Junio) Landmark. [Online]. https://www.landmark.solutions/SeisSpace-ProMAX

[2] Halliburton|Landmark. (2016, Enero) Landmark. [Online]. https://www.landmark.solutions/About

[3] A Tarantola, "Inversion of seismic-reflection data in the acousticapproximation," *Geophysics*, vol. 49, no. 8, pp. 1259-1266, 1984.

[4] David Leonardo Abreo, "Implementación de una inversión de onda completa (FWI) 2-D sobre una arquitectura," Escuela de Ingenieria Electrica, Electronica y Telecomunicaciones, Universidad Industrial de Santander, Bucaramanga, Tesis pregrado 2015.

[5] S. A Abreo Carrillo, A Ramirez, O, M Reyes, D. L Abreo Carrillo, and H. & González Alvarez, "A practical implementation of acoustic full waveform inversion on graphical processing units," *CT&F - Ciencia, Tecnología y Futuro*, vol. 6, no. 2, pp. 5-19, 2015.

[6] R Plessix, "A review of the adjoint-state method for computingthe gradient of a functional with geophysical applications," *Geophysical Journal International* , vol. 167, no. 2, pp. 495-503, 206.

[7] Halliburton|Landmark Software & Services, "ProMAX® Developer's Programming Guide Contents," in *ProMAX® and SeisSpace® Online Documents*. Texas, 2010.

[8] Jorge Monsegny, "Seismic proceing module developing for ProMAX/SeisSpace," Universidad Industrial de Santander, Bucaramanga, Capacitación 2015.

[9] R Stallman, R McGrath, and P Smith, "GNU Make: A Program for Directing Recompilation," Boston, Manual 2014.

[10] Antonio Luque Estepa, "Herramientas de desarrollo bajo Linux," Escuela

Superior de Ingenieros de Sevilla, España, 2000.

[11] BYRON and GOTTFRIED, *Programación en C*, Segunda edición ed. Mexico: Mc Graw Hill, 2015.

[12] Gonzalo Ávila. (2006, Enero) Gzalo.com. [Online]. http://gzalo.com/compiladores_gnu/

[13] J Virieux and S Operto, "An overview of full-waveform inversionin exploration geophysics," *Geophysics*, vol. 74, no. 6, pp. WCC1-WCC26, 2009.

BIBLIOGRAFÍA

A Tarantola, "Inversion of seismic-reflection data in the acousticapproximation," *Geophysics*, vol. 49, no. 8, pp. 1259-1266, 1984.

Antonio Luque Estepa, "Herramientas de desarrollo bajo Linux," Escuela Superior de Ingenieros de Sevilla, España, 2000.

BYRON and GOTTFRIED, *Programación en C*, Segunda edición ed. Mexico: Mc Graw Hill, 2015.

David Leonardo Abreo, "Implementación de una inversión de onda completa (FWI) 2-D sobre una arquitectura," Escuela de Ingenieria Electrica, Electronica y Telecomunicaciones, Universidad Industrial de Santander, Bucaramanga, Tesis pregrado 2015.

Gonzalo Ávila. (2006, Enero) Gzalo.com. [Online]. http://gzalo.com/compiladores_gnu/

Halliburton|Landmark Software & Services, "ProMAX® Developer's Programming Guide Contents," in *ProMAX® and SeisSpace® Online Documents*. Texas, 2010.

Halliburton|Landmark. (2015, Junio) Landmark. [Online]. https://www.landmark.solutions/SeisSpace-ProMAX

Halliburton|Landmark. (2016, Enero) Landmark. [Online]. https://www.landmark.solutions/About

J Virieux and S Operto, "An overview of full-waveform inversionin exploration geophysics," *Geophysics*, vol. 74, no. 6, pp. WCC1-WCC26, 2009.

Jorge Monsegny, "Seismic proceing module developing for ProMAX/SeisSpace," Universidad Industrial de Santander, Bucaramanga, Capacitación 2015.

S. A Abreo Carrillo, A Ramirez, O, M Reyes, D. L Abreo Carrillo, and H. & González Alvarez, "A practical implementation of acoustic full waveform inversion on graphical processing units," *CT&F - Ciencia, Tecnología y Futuro*, vol. 6, no. 2, pp. 5-19, 2015.

R Plessix, "A review of the adjoint-state method for computingthe gradient of a functional with geophysical applications," *Geophysical Journal International* , vol. 167, no. 2, pp. 495-503, 206.

R Stallman, R McGrath, and P Smith, "GNU Make: A Program for Directing Recompilation," Boston, Manual 2014.