UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

LICENCIATURA EN TECNOLOGÍAS PARA LA INFORMACIÓN EN CIENCIAS

Escuela Nacional de Estudios Superiores, Unidad Morelia

PARALELIZACIÓN DE ALGORTIMOS DE BÚSQUEDA DE SISMOS REPETITIVOS UTILIZANDO UNIDADES DE PROCESO GRÁFICO (GPU)

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN TECNOLOGÍAS PARA LA INFORMACIÓN EN CIENCIAS

P R E S E N T A

GERARDO ALBERTO RODRÍGUEZ VALENCIA

DIRECTOR(A) DE TESIS: DR. LUIS ANTONIO DOMÍNGUEZ RAMÍREZ

MORELIA, MICHOACÁN **FEBRERO, 2020**

ESCUELA NACIONAL de ESTUDIOS SUPERIORES UNIDAD MORELIA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
ESCUELA NACIONAL DE ESTUDIOS SUPERIORES, UNIDAD MORELIA
SECRETARÍA GENERAL
SERVICIOS ESCOLARES

MTRA. IVONNE RAMÍREZ WENCE
DIRECTORA
DIRECCIÓN GENERAL DE ADMINISTRACIÓN ESCOLAR
P R E S E N T E

Por medio de la presente me permito informar a usted que en la **sesión extraordinaria 10** del **H. Consejo Técnico** de la Escuela Nacional de Estudios Superiores (ENES) Unidad Morelia celebrada el día **29 de mayo del 2019**, acordó poner a su consideración el siguiente jurado para la presentación del Trabajo Profesional del alumno (a) **Gerardo Alberto Rodríguez Valencia** de la Licenciatura en **Tecnologías para la Información en Ciencias,** con número de cuenta **415019938**, con la tesis titulada: "Paralelización de algoritmos de búsqueda de sismos repetitivos utilizando unidades de procesamiento gráfico (GPU)", bajo la dirección como **tutor** del Dr. Luis Antonio Domínguez Ramírez y como **co-tutora** la Dra. Adriana Menchaca Méndez.

El jurado queda integrado de la siguiente manera:

Presidente:	Dra. Marisol Flores Garrido
Vocal:	Dr. Víctor Hugo De la Luz Rodríguez
Secretario:	Dr. Luis Antonio Domínguez Ramírez
Suplente 1:	Dra. Miriam Pescador Rojas
Suplente 2:	Dr. Luis Miguel García Velázquez

Sin otro particular, quedo de usted.

A t e n t a m e n t e
"POR MI RAZA HABLARA EL ESPIRITU"
Morelia, Michoacán a 27 de febrero del 2020.

DR. VÍCTOR HUGO ANAYA MUÑOZ
SECRETARIO GENERAL

Agradecimientos institucionales

Agradezco a:
La Escuela Nacional de Estudios Superiores Unidad Morelia, y a la Universidad Nacional Autónoma de México, por haberme permitido formar parte de su institución y brindarme su apoyo y enseñanzas.

Alejandro Rebollar y el cuerpo de servicios escolares.

La beca del proyecto UCMEXUS 1732 "Discovering the connection between slow slip events and large megathrust earthquakes in Mexico", por haberme dado el apoyo económico.

Al proyecto TAMU 2017-027 "Investigation of slip behavior and seismic hazard along the Mexican subduction zone".

Mis profesores de la carrera: Marisol Flores, Miguel Raggi, Daniele Colosi, Andrés Torres, Víctor Anaya, Sergio Tinoco, Adriana Menchaca, Luis Miguel, quienes me brindaron su tiempo y su conocimiento para mi formación.

Luis Antonio Domínguez, por haber fungido como mi asesor y haberme aceptado en el proyecto para adquirir nuevos conocimientos y habilidades.

Adriana Menchaca, por su apoyo en el transcurso de la carrera y su apoyo como co-asesora.

Mis sinodales por dar su tiempo para este proyecto:
Dra. Marisol Flores Garrido
Dr. Luis Miguel García Velázquez
Dr. Victor De la Luz
Dra. Miriam Pescador Rojas

Agradecimientos personales

Agradezco a:
Mi familia, por todo el apoyo y consejos que me ha brindado en todos estos años; mis padres Mario y Regina, quienes siempre estuvieron para mí, y a mis hermanos Aldo y Karla, quienes siempre me dieron sus consejos y experiencias.

Mis amigos:
Emily Sol, por brindarme siempre apoyo emocional.
El primer egresado de materiales, Luis "Gigis"Zaldivar.
Itzel "Oaxaca", por escucharme y estar para mi siempre y en todo.
Memo Suazo. La escuela es temporal, pero la lucha por el país es para siempre.
Denise, por estar cuando lo necesite.

Todas las demás personas que conocí en mis años universitarios y que me brindaron de su apoyo y amistad, pues aunque no está de más mencionarlas, son demasiadas para enlistar.

Resumen

En años recientes, el Servicio Sismológico Nacional ha aumentado significativamente su capacidad de monitoreo y registro sísmico en México. Esto ha permitido la detección de un número cada vez mayor de sismos en todo el país. Gracias a la instalación de nuevas estaciones sísmicas, es posible conocer con mayor detalle las zonas sísmicas de México y su relación con la generación de grandes sismos. Sin embargo, esto requiere del procesamiento de un volumen cada vez mayor de datos, y por ende, del desarrollo de herramientas y algoritmos de cómputo capaces de analizar en tiempos relativamente cortos la información proporcionada.

Aunque existe una gran variedad de fenómenos sísmicos, un tipo de evento sísmico de particular interés para este proyecto son los sismos repetitivos. Este tipo de sismos nos permite estimar la tasa de deslizamiento entre las placas a profundidades de varios kilómetros, donde no es posible la instrumentación directa. La característica principal que los define es que la localización y la zona de ruptura generan formas de onda casi idénticas en las estaciones que los detectan; por lo cual, se les asocia con asperezas en la interfaz de las placas, las cuales almacenan y liberan energía de forma regular.

Estos sismos representan menos del 0.1 % y para identificarlos es necesario evaluar cada uno y compararlo con todos los sismos previos ocurridos en la misma zona para determinar si se trata de un sismo repetitivo o no. Para llevarlo a cabo es necesario el cálculo de los coeficientes de correlaciones y de coherencia espectral para determinar la similitud en la forma de onda y, consecuentemente, determinar si es un sismo repetitivo.

En este proyecto se busca disminuir drásticamente el tiempo necesario de cómputo para la detección de sismos repetitivos mediante el uso de tarjetas gráficas o GPU. A diferencia del cómputo tradicional, las tarjetas gráficas permiten el procesamiento de datos en paralelo de manera rápida y eficiente ya que todos las unidades de proceso se encuentran físicamente en una misma unidad. De esta forma es posible analizar

un gran volumen de señales sísmicas en paralelo, reduciendo en varios órdenes de magnitud el tiempo necesario para el procesamiento de los datos.

Abstract

In recent years, the National Seismological Service has significantly increased its capacity for monitoring and seismic recording in Mexico. This has allowed the detection of an increasing number of earthquakes throughout the country. Thanks to the installation of new seismic stations, is possible to know in greater detail the seismic zones of Mexico and their relation to the generation of large earthquakes. Nonetheless, this requires the processing of an increasing volume of data, the development of efficient computing tools and algorithms capable of analyzing the information provided in relatively short times.

Although there is a great variety of seismic phenomena, a type of seismic event of particular interest for this work, are the so-called repetitive earthquakes. This type of earthquake allow us to estimate the slip rate between the plates, at depths of several kilometers where direct instrumentation is not possible. The main characteristic that defines them is that the location and zone of rupture generate almost identical waveforms in the stations that detect them, which is why they are associated with asperities in the interface of the plates which store and release energy in a regular way.

These earthquakes represent less than 0.1 %, so to identify them it is necessary to evaluate each earthquake and compare it with all previous ones in the same zone to determine if it is a repetitive earthquake or not. For this it is necessary to calculate the coefficients of correlations and spectral coherence to determine the similarity in the waveform and consequently to determine if is an repetitive earthquake.

In this project, we drastically reduce the necessary computation time for the detection of repeting earthquakes through the use of graphic cards or GPUs. Unlike traditional computing, graphic cards allow the parallel processing of data quickly

and efficiently since all the process units are physically in the same unit. In this way it is possible to process a large volume of seismic signals in parallel, reducing the time required for data processing by several orders of magnitude.

Índice general

CAPÍTULO 1

INTRODUCCIÓN

Contenido

La rápida evolución de las tecnologías de cómputo ha permitido que cada vez sea más fácil y accesible el procesamiento de grandes volúmenes de datos. Una de las tecnologías que ha tomado relevancia en el mundo científico, en la última década, son las unidades de proceso gráfico (GPU por sus siglas en inglés). Este tipo de dispositivos fueron incorporados recientemente en las computadoras comerciales de uso general para el manejo de ambientes gráficos, utilizados principalmente por la industria de los videojuegos. La comunidad científica ha adoptado esta tecnología para el procesamiento en paralelo de grandes volúmenes de información y para la simulación de diversos problemas físicos, químicos y biológicos. El procesamiento de datos en GPU ofrece múltiples ventajas con respecto a otros paradigmas de programación científica como el cómputo distribuido o de multiprocesadores. Algunas de las ventajas son:

- Utilización de una arquitectura unificada. A diferencia del cómputo distribuido, el procesamiento de datos en GPU permite el acceso a las unidades aritmético lógicas y de memoria localizadas dentro de un módulo. Esto reduce problemas de compatibilidad entre diferentes sistemas, lo que facilita considerablemente la programación.

- Bajo nivel de latencia causada por la transferencia de datos de un punto a otro. Una vez que son enviados los datos de las unidades de almacenamiento (RAM, discos duros, etc.) de la unidad central de proceso a la GPU, los tiempos de transferencia de datos se reducen considerablemente ya que todos los elementos de proceso se encuentran físicamente cercanos entre sí.
- Flexibilidad en la programación. Otra ventaja importante es la facilidad que existe para paralelizar procesos y/o utilizar diferentes equipos de cómputo siempre y cuando se utilicen procesadores gráficos de un mismo fabricante.

1.1. Motivación

Uno de los aspectos más relevantes de la evolución tecnológica es el aumento en la capacidad de almacenamiento de datos. Por ejemplo, en tan sólo una década, las computadoras personales pasaron de utilizar discos magnéticos con una capacidad de 1.44 MB, a unidades de CD de aproximadamente 700 MB, a unidades USB con capacidades mayores a 1 GB como unidades de memoria removibles. Este aumento en la capacidad de almacenamiento ha permitido que las redes de monitoreo sísmico tengan registros en tiempo continuo que hasta hace poco tiempo hubiera sido impensable. En nuestro caso, utilizaremos las GPU para el procesamiento de información proveniente de la Red Sismológica Nacional. Esta red, como muchas otras redes del mundo, ha tenido una transformación importante gracias a la capacidad de almacenamiento. Previo al desarrollo de los dispositivos de almacenamiento masivo de datos. El Servicio Sismológico Nacional (SSN) registraba sismos en modo disparo, es decir, registraba unos cuantos segundos de información una vez detectado un movimiento del subsuelo. Actualmente, las estaciones sísmicas de la red permanente del SSN almacenan los registros sísmicos de forma continua, las 24 horas del día, los 365(6) días del año en un tasa típica de muestreo de 100 muestras por segundo. La Figura 1.2 muestra el número de sismos detectados al año por el SSN. Como se puede ver, existe un crecimiento exponencial en el número de sismos detectados que es resultado tanto de la instalación de nuevas estaciones sísmicas, como de la evolu-

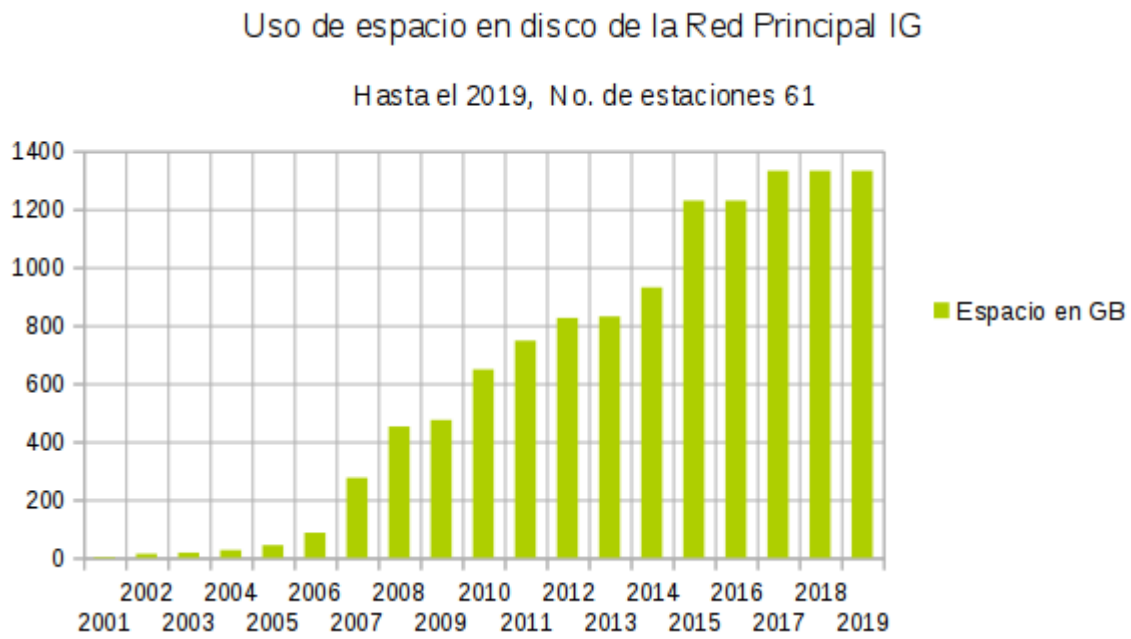

Figura 1.1: Almacenamiento reportado por el SSN del 2001 al 2019. Al 2019, el SSN recibe datos de manera continua de 61 estaciones permanentes, cada una de las cuales registra 6 canales de datos (3 canales de velocidad y 3 de aceleración). Servicio Sismológico Nacional, UNAM; Enero 2020

ción de los algoritmos de detección. En la Figura 1.1 se muestra como ha aumentado el volumen de datos registrados por el SSN, de unos cuantos GB en 2001 a cerca de 1400 GB por año en 2019. Por ende, es necesario la creación y optimización de algoritmos que se adapten tanto a la cantidad de datos que se crean diariamente, como a la capacidad de cómputo donde se ejecutarán. El objetivo general de este trabajo de tesis es usar la capacidad de las nuevas tecnologías para procesar una cantidad considerable de datos en el menor tiempo posible.

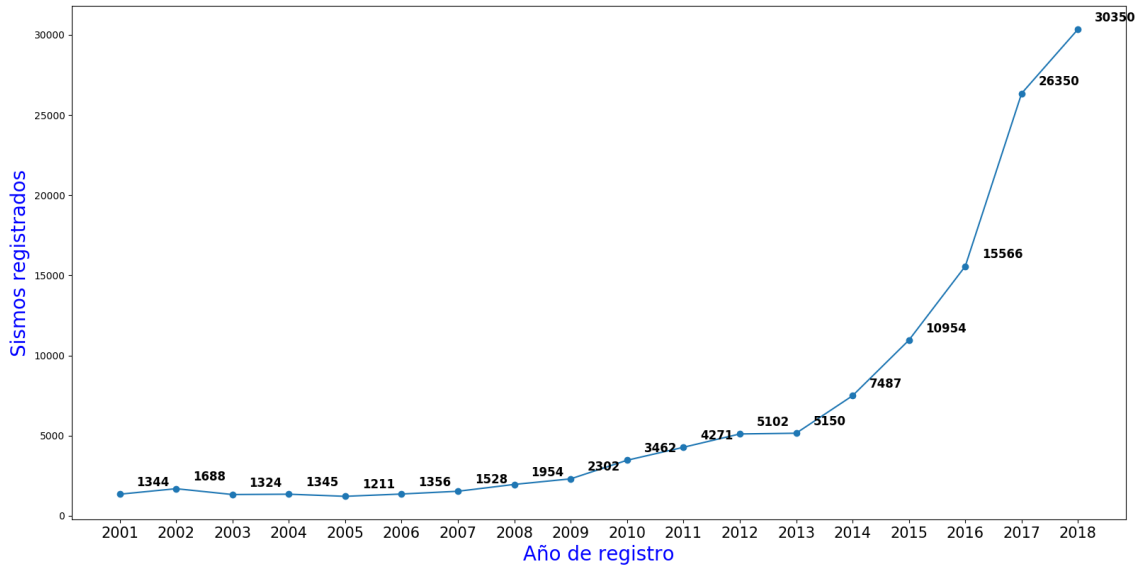

Figura 1.2: La cantidad de sismos registrados por el SSN en las últimas dos décadas va en un aumento exponencial, como resultado de la cantidad de estaciones utilizadas. Esto no implica que haya habido un incremento en la cantidad de sismos que se generan por año, pero sí en la cantidad que se registra. Datos extraídos de la base de datos del Servicio Sismológico Nacional.

1.2. Objetivo

El objetivo de este proyecto es paralelizar un algoritmo para el procesamiento de señales sísmicas a partir del análisis espectral y de herramientas diseñadas para cálculos en GPU.

1.2.1. Objetivos específicos

- Paralelizar un algoritmo de búsqueda e identificación de sismos repetitivos, mediante el cómputo paralelo.
- Calcular los coeficientes de correlación y coherencia espectral de pares de sismos registrados por el Servicio Sismológico Nacional de 2001-2018.

1.3. Descripción general

A continuación se presenta una síntesis de cada uno de los capítulos que componen esta tesis y su importancia en el desarrollo de la misma:

- En el **capítulo 2** se menciona la importancia de generar un código que esté bien estructurado y documentado. En particular, se busca generar un código que sea modular, expansible y reutilizable para trabajos futuros que permitan la colaboración entre distintos programadores, así como una futura expansión y mejora. Además, se hace énfasis en la representación numérica de los datos. Esto es de especial importancia para la presente tesis, ya que una mala representación numérica puede llevar a un consumo innecesario de memoria y, por lo tanto, un incremento significativo en los tiempos de ejecución.

- En el **capítulo 3** se hace una breve descripción de los tipos de arquitecturas desarrollados por la compañía NVIDIA, principal fabricante de las GPU, tanto de uso recreativo (videojuegos) como científico. Además, se muestra la rápida evolución de esta industria y su impacto en el cómputo de alto rendimiento. Finalmente, se introducen los conceptos básicos del lenguaje de programación CUDA, que fue el lenguaje utilizado para el procesamiento y análisis de los datos de este trabajo de tesis.

- En el **capítulo 4** se introducen los conceptos básicos de la sismología y de la actividad tectónica en México. Asimismo, se hace una descripción general de los sismos repetitivos, los cuales son el tema de estudio principal que se intenta mejorar a través de esta tesis.

- El **capítulo 5** ofrece una revisión de los conceptos de procesamiento de señales utilizados en este trabajo. En particular, se utiliza la Transformada de Fourier para pasar de una señal en el dominio del tiempo a una señal en el dominio de frecuencia. Esto permite el tratamiento de la señal de una forma eficiente a través del uso de librerías especializadas en el cálculo de la Transformada de Fourier.

- El **capítulo 6** muestra el desarrollo del código utilizado y de los diferentes pasos en el procesamiento de las señales. En este capítulo se describen los principales componentes del código y su implementación en CUDA.

- El **capítulo 7** muestra las pruebas de rendimiento realizadas al código. En particular, se muestra la reducción en los tiempos de cómputo obtenidos al utilizar una GPU. Además, se comparan los resultados numéricos de los cálculos utilizando C y CUDA.

- Finalmente, en el **capítulo 8** se hace un resumen general de los resultados obtenidos y del trabajo futuro que pudiera realizarse como consecuencia de esta tesis.

CAPÍTULO 2

MARCO TEÓRICO

Contenido

Las tecnologías de la información son una de las ramas de la ciencia que mayor impacto tiene en la vida di aria de las personas y en el desarrollo científico. Si bien es cierto que cada día surgen nuevos dispositivos, algoritmos y paradigmas de programación, existen conceptos básicos que son comunes a todos ellos. En otras palabras, independientemente de las herramientas que se decidan utilizar para resolver un problema en específico, existen principios que debemos de considerar antes y durante el desarrollo de nuestra aplicación.

En este capítulo se establecen una serie de normas que creemos se deben de satisfacer para que el software que se desarrolle cumpla no solo con los requerimientos del problema, sino que también permita un desarrollo, revisión y ampliación del mismo de manera rápida, ordenada y bien estructurada. Adicionalmente, se describen los tipos de datos utilizados para el almacenamiento en memoria de la información.

Entender la representación numérica es de suma importancia para el problema que buscamos resolver, ya que nuestro objetivo es el procesado de un gran volumen de datos, no hacerlo nos puede llevar a la sobre utilización de recursos de memoria. Finalmente, se hace una breve revisión de las técnicas de procesamiento en paralelo.

2.1. Filosofía de la programación

Una de las metas principales en el desarrollo de nuevo software científico es optimizar un proceso mediante el procesamiento y el análisis de grandes cantidades de datos que se generan en proyectos de investigación. Por lo que el software científico necesita ser creado de manera tal que sea fácil de comprender tanto por la persona que lo escribe como por otros programadores en un ambiente colaborativo de trabajo. Bajo esta premisa, es necesario seguir un conjunto de reglas que permitan que el código que creamos sea escalable, y pueda ser probado regularmente de forma que los cambios introducidos por el programador principal, o por alguno de los colaboradores produzca el resultado esperado sin introducir vulnerabilidades en el mismo.

2.2. Características de un buen código

Las computadoras realizan una gran cantidad de procesos internos, que deben de ser considerados al momento de escribir, compilar y ejecutar un código para asegurar su correcto funcionamiento. Por ejemplo, si utilizamos un lenguaje de programación de relativo bajo nivel (como los que piensa utilizar en esta tesis), es posible que el algoritmo actualice y/o acceda a los contenidos de una misma dirección de memoria, lo cual puede introducir errores lógicos sin ser advertidos por el compilador. Por lo que se debe de tener especial cuidado en la asignación, uso y liberación de los blo-

ques de memoria, especialmente cuando se utiliza memoria dinámica[1]. Además, se debe de considerar la reutilización del código a través del uso de funciones que haga que nuestro código no sólo más compacto, sino también disminuya la posibilidad de introducir errores al momento de reescribir funciones de forma recursiva. Finalmente, teniendo en cuenta que cada actualización del código puede introducir nuevos errores, deberíamos de realizar pruebas al código que estamos usando. Una forma de verificación es la unidad de prueba (*unit test*), la cual consiste en asegurarse que una parte de código regresa resultados correctos o verificar que el comportamiento del programa no cambia cuando se modifica el código. Podemos integrar una biblioteca de pruebas, esto para que los programadores diseñen y construyan código comprobable, es decir, crear funciones autónomas que se ejecuten lo más independientemente posible.

Otra manera para propiciar la buena ejecución y uso del software por otro usuario es la documentación, en la cual se detalla el funcionamiento del software. Un código bien documentado hace que sea más fácil entender no solo el código, sino el proceso que representa. Podemos decir que el tiempo que se invierte en la creación del software es proporcional a la satisfacción de los resultados entregados.

2.3. Representación numérica

La base sobre la cual opera un sistema numérico corresponde al número de símbolos que se utilizan para representar una cantidad. Por ejemplo, la base 10 usa diez símbolos para expresarse y la base 2 usa únicamente dos. Para poder diferenciar estas bases, generalmente se usa N_b, donde N es nuestra cifra y b es la base en la que está escrita. Por ejemplo, 10_2 indica que el número 10 está en base 2. Cuando un número no tiene esta notación, se asume que está en base 10 (nuestro sistema decimal). Aunque la representación en la interfaz con el usuario de una computadora

[1] La memoria dinámica es aquella memoria cuyo tamaño se asigna durante la ejecución del código

es base 10, internamente la información se almacena de forma binaria.
Un aspecto importante a considerar cuando se almacena información de forma binaria es el concepto de *Endianness*. Este término se refiere a como se ordenan los bits en memoria de una palabra. Lo que conlleva a elegir si se guarda el byte menos significativo en la dirección de memoria más baja o el byte más significativo en la dirección de memoria más baja. De aquí podemos diferenciar, entonces, el *big endian*, que es la manera de guardar el byte más significativo primero y el *little endian*, que es la manera de guardar el byte menos significativo primero. Ver Tablas 2.1 y 2.2. En sismología, este es un problema relativamente común de encontrar debido a la diversidad de fabricantes y edad de los equipos. Sin embargo, actualmente la mayoría de las librerías de procesamiento de datos sísmicos son capaces de leer archivos en cualquiera de estos formatos.

0D	0C	0B	0A

Tabla 2.1: *Big endian*

0A	0B	0C	0D

Tabla 2.2: *Little endian*

2.3.1. Binario

El sistema de base 2 es generalmente usado para interpretar los demás sistemas numéricos en una computadora. Las reglas para poder representar un número en este sistema son las siguientes: se debe tomar en cuenta la posición de cada 1 en el arreglo, donde el 1 representa el valor de la posición donde se encuentra. Cada posición del arreglo tiene un valor único, siendo el valor calculado mediante 2^n, donde n representa la posición del 1 contando de derecha a izquierda empezando por cero. Por ejemplo, el número binario 100110, donde, en el arreglo, las posiciones tienen el valor $2^5, 2^4, 2^3, 2^2, 2^1, 2^0$ que es lo mismo que $32, 16, 8, 4, 2, 1$. Sumamos los valores donde se encuentra posicionado el 1 en el número binario $32 + 4 + 2$. Por lo tanto, el número binario 100110 representa al 38 en el sistema decimal.

2.3.2. Valores enteros

En una computadora de 32 bits, los valores tipo entero se fijan en 4 bytes (32 espacios para representar un número en binario) para su almacenamiento en la memoria. Lo suficiente para contener cualquier número en el rango de -2,147,483,648 a 2,147,483,647. Existen dos casos, la representación del número con signo (positivo o negativo) y la representación sin signo que se asume es positivo. Para los valores sin signo, el valor binario del número se representa con b^n, siendo b el tipo de base, en este caso 2, y n la cantidad de bits de la palabra. Recordemos que el entero puede usar 4 bytes, es decir, 32 bits (64 alcanzables actualmente). La forma de decodificar el código binario a un entero sin signo es prácticamente el mismo proceso que el explicado en la Subsección anterior. La diferencia se produce cuando un entero tiene un signo, para lo cual el proceso varía como sigue:

El siguiente binario tiene un valor decimal de +1100

0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 1 0 0 1 1 0 0

Mientras que el siguiente binario tiene un valor decimal de -1100

1	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 1 0 0 1 1 0 0

2.3.3. Valores reales

Para este tipo de valores se toma el lugar donde el punto separa la parte decimal y la fraccionaria, esto es el punto fijo, en el cual el punto decimal está fijo dentro de la cifra (tienen el signo, que definirá si la cifra es negativa o positiva.), se elegirá el lugar arbitrariamente y no se podrá cambiar la posición del punto.

El espacio que tomará en memoria está distribuido en diferentes partes, el signo tendrá 1 bit, el exponente tendrá 8 bits y la mantisa tendrá 23 bits. El proceso de decodificación será el mismo que se usa para los números enteros, para la parte

decimal, la fórmula 2^n, donde n es la posición. En la parte fraccionaria, después del punto, n cambia sus valores a negativos, es decir, $2^{-1}, 2^{-2}, 2^{-3}$... Por ejemplo:

$$1.101 = 2^0.2^{-1} + 2^{-3}$$

Para el cálculo del punto flotante, el punto puede variar en la posición de la cifra, suele usarse la normalización de la cifra binaria, esto se hace moviendo el punto y dejando en la parte entera un solo dígito diferente de 0. Por ejemplo, el binario 10110.101 se normaliza 1.0110101x10^4. El IEEE (*Institute of Electrical and Electronics Engineers*) tiene tres formatos para los puntos flotantes, de los cuales definen un espacio de memoria diferente para cada tipo de punto flotante. Para precisión simple se asigna un total de 32 bits, distribuidos como se explicó anteriormente. Para precisión doble se usan 64 bits, donde se distribuye 1 bit para el signo, 11 para el exponente y 52 para la mantisa. Finalmente, 128 bits para precisión cuádruple[2], 1 bit para el signo, 15 bits para el exponente y 112 bits para la mantisa.

2.4. Valores hexadecimales (Hex)

Este tipo de valores posicionales son números en base 16. Cada posición del arreglo puede tomar cualquiera de los siguientes dieciséis valores: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Es usado comúnmente en los sistemas digitales para la reducción de cadenas de números binarios en conjuntos de cuatro dígitos. Cada dígito del número tiene una base 16, a partir del bit menos significativo. Los programadores suelen usar números hexadecimales en lenguajes de programación tipo ensamblador, es decir lenguajes de bajo nivel donde se programan instrucciones directo en el procesador y se tiene acceso directo a las direcciones de memoria.

[2]Llamada comúnmente cómo precisión quad, del inglés *Quadruple precision* o *Quad precision*

Para la conversión a otros sistemas, se toman los valores de A, B, C, D, E y F como si fueran los números decimales que siguen a partir de la posición que representan en el rango, es decir, 10, 11, 12, 13, 14 y 15, respectivamente. Entonces se usa el mismo método de conversión que en binario. Por ejemplo, si tenemos el valor hexadecimal F3A, tendría los siguientes valores binarios correspondientes. F = 1111, 3 = 0011, A = 1010, luego el resultado sería la concatenación de los valores en binario, 111100111010.

Para la conversión de hexadecimal a decimal, transformamos cada dígito del número hexadecimal a su correspondiente número decimal. Después, cada valor se multiplica por 16^n, donde n representa la posición del valor de derecha a izquierda, para obtener como resultado una suma de las multiplicaciones. Por ejemplo, para el valor hexadecimal F12A4 obtenemos el valor decimal de cada dígito: F = 15, 1 = 1, 2 = 2, A = 10, 4 = 4. Posteriormente, se multiplican por su 16^n según su posición y se suman.
$15*16^4+1*16^3+2*16^2+10*16^1+4*16^0 = 983040+4096+512+160+4 = 987812$

En la Tabla 2.3 se pueden observar otros tipos de valores que son usados comúnmente. Entender estos tipos de datos es esencial cuando se trabaja con grandes volúmenes de datos, o cuando se requiere leer un archivo binario que contiene diferentes tipos de datos. Éste es el caso de los archivos SAC que contienen la información de datos sísmicos y de cuales hablaremos más adelante.

2.5. Programación en paralelo

Para lograr los objetivos de este proyecto es necesario comprender las técnicas que se utilizarán. El algoritmo que se usa se basa en la paralelización de procesos. La programación en paralelo consiste de una técnica de cómputo para la cual se ejecutan instrucciones de manera simultánea, dividiendo así el problema y la carga de trabajo en partes más pequeñas. A diferencia de los programas secuenciales, los

Tipo de variable	Número de bits	Rango de valores
char	8	-128 a 127 o 0 a 255
unsigned char	8	0 a 255
signed char	8	-128 a 127
short	16	32,768 a 32,767
int	16	-32,768 a 32,767 o -2,147,483,648 a 2,147,483,647
Unsigned int	16	0 a 65,535 o 0 a 4,294,967,295
short int	16	-32,768 a 32,767
long	32	-2,147,483,648 a 2,147,483,647
long int	32	-2,147,483,648 a 2,147,483,647
float	32	3.4E +/- 38 (7 dígitos)
double	64	1.7E +/- 308 (15 dígitos)

Tabla 2.3: Rangos de valores para algunos tipos de dato.

cuales ejecutan procesos o instrucciones donde cada instrucción pasa una a la vez a ejecución en la memoria del CPU y en cuanto termina su ejecución, inmediatamente otra entra. El cómputo paralelo utiliza varias unidades centrales de proceso (CPU) que pueden estar concentradas en un solo clúster o súper computadora, distribuido en distintos equipos con diferentes sistemas operativos intercomunicado a través de una misma red. [Reinders, 2007] y [Hyyrö et al., 2005] describen diferentes tipos de paralelismo:

- **Paralelismo a nivel instrucción**. Es una combinación de instrucciones de bajo nivel que el procesador ejecuta al mismo tiempo sin afectar el resultado. Ver Tabla 2.4.
- **Paralelismo a nivel de hilo de ejecución**. Aquí diferentes hilos [3] comparten unidades del procesador y este debe de tener una estructura independiente para cada hilo.
- **Paralelismo a nivel bit**. Una palabra necesita cierta cantidad de bits de ejecución. Mientras la palabra sea más grande, menos instrucciones se necesitan

[3] Un hilo es una abstracción de un proceso que será ejecutado en el kernel.

Ciclos / Iteraciones	1	2	3	4	5	6	7	8	9
Instrucción 1	IF	ID	EX	MEM	WB				
Instrucción 2		IF	ID	EX	MEM	WB			
Instrucción 3			IF	ID	EX	MEM	WB		

Tabla 2.4: Ejemplo de paralelismo a nivel instrucción. Se tienen tres instrucciones que se ejecutarán una después de otra tras terminar una etapa de la instrucción que se ejecutó antes. En el ejemplo podemos ver la primera etapa de procesamiento de la instrucción 1, *Fetch* (IF), se ejecuta en la instrucción 1 durante el primer ciclo de reloj. Durante el segundo ciclo de reloj se ejecuta la segunda etapa de instrucción 1, *Decode* (ID), y la primera etapa de instrucción 2, *Fetch* (IF), en el tercer ciclo de reloj, se ejecuta la tercera etapa de la instrucción 1, *Execute* (EX), la segunda etapa de la instrucción 2, *Decode* (ID), y la tercera etapa de la instrucción 3, *Execute* (EX), así hasta completar todas las etapas de cada instrucción.

para su ejecución. Esta técnica consiste en aumentar el tamaño de las palabras para disminuir la cantidad de instrucciones para su ejecución en el procesador.

- **Paralelismo a nivel datos**. Trata de distribuir diferentes datos en varios procesadores para ser ejecutados en paralelo y se tenga un solo resultado.
- **Paralelismo a nivel tareas**. Diferentes cálculos que se ejecutan en diferentes conjuntos de datos o un mismo conjunto. Es parecido al paralelismo a nivel de datos.

Una de las dificultades que enfrenta el paralelismo es la comunicación entre las subtareas ya que el rendimiento se ve afectado por la falta de comunicación o la interrupción de los procesos. Además escribir un programa de paralelización es más complejo que escribir un programa secuencial.

CAPÍTULO 3

TARJETAS DE PROCESAMIENTO GRÁFICO

Contenido

El origen de las GPU (Unidad de Procesamiento Gráfico, del inglés *Graphics Processing Unit*) surge por la necesidad de optimizar y mejorar los gráficos en computadora para brindar una experiencia más realista, como en aplicaciones de videojuegos y simulaciones [Francesc Guim, 2014]. La solución más pronta que se encontró para este problema, fue aumentar el número de unidades de procesadores, por lo que esto trajo una mayor velocidad y la creación de un microprocesador que funcionara independiente del CPU. Es decir, ejecutara procesos por su propia cuenta liberando recursos del CPU. Con el paso del tiempo, los procesos que podía manejar este microprocesador eran cada vez mayores, independizandose cada vez más del

CPU. Las operaciones que podía ejecutar inicialmente eran el filtrado de texturas y geometría del entorno. En 1999, la corporación NVIDIA presentó su microprocesador GeForce 256 nombrándolo por primera vez como GPU y añadiendo funciones nuevas como transformación e iluminación.

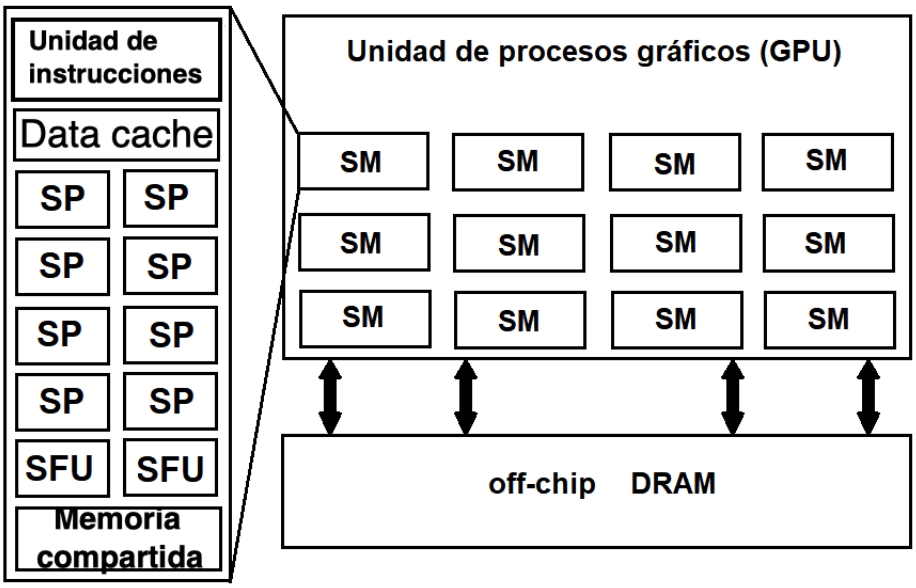

Figura 3.1: Arquitectura de una GPU. En la memoria DRAM (*Dynamic RAM*), *off-chip* significa que los componentes no están integrados al chip, al contrario de *on-chip*, como pueden ser los transistores. Esto tiene relevancia en la latencia de respuesta de los elementos. La imagen muestra cómo se divide una GPU y las partes que componen a un *streaming multiprocessor*. [Gebremedhin et al., 2012].

Una característica especial de las GPU es su estructura paralelizada, la cual nos permite manipular grandes cantidades de información de manera eficiente. Cada GPU está compuesta por varias unidades de procesamiento, Streaming Multiprocesor (SM), que representan el primer nivel lógico de paralelismo. La GPU está conectada a la memoria DRAM, diferente de la memoria SRAM [1]. A su vez, cada SM se divide en Stream Processors (SP), ver Figura 3.1. Cada SP tiene un núcleo de

[1] *Static RAM, mantiene los datos con mayor uso en la memoria*

ejecución que puede correr hilos secuencialmente (un hilo es una abstracción que representa la ejecución de un proceso). Además, cada SM tiene un número de registros, estos representan la parte de la memoria de acceso rápido que es temporal y limitada en tamaño. La GPU accede a una memoria llamada memoria VRAM, que es del mismo tipo que la memoria RAM. Esta memoria es usada para cálculos, gráficos y sirve como ayuda para el procesamiento de las primitivas (operaciones gráficas para el proceso de píxeles). De igual manera, podemos dividir la memoria en dos: dedicada y compartida. La memoria dedicada no es más que la cantidad de memoria reservada para la tarjeta gráfica y los procesos que la utilizarán, mientras que la memoria compartida es la cantidad de memoria que puede ser compartida por varios programas para su ejecución.

Los procesos que ejecuta una GPU funcionan mediante un *pipeline*, esto es, una serie de elementos para procesar, organizados y especializados en funciones que se ejecutan en cierto orden definido. El pipeline está compuesto por etapas, en las cuales cada una recibe la salida de una etapa anterior y proporciona una salida para el siguiente *pipeline*. El proceso de ejecución de una instrucción se puede describir en el siguiente orden de fases:

1. Fetch (IF): Busca en la memoria la instrucción que se ejecutará, la cual es identificada mediante un contador que aumenta en uno y apunta a la localización de la siguiente instrucción.
2. Decode (ID): Con una instrucción como salida de la etapa Fetch, la unidad de control decodifica la instrucción usando una porción de su código que se encargará de especificar qué operación se le aplicará a la instrucción.
3. Execute (EX): Procesada por la unidad aritmética lógica (ALU, por sus siglas en inglés), realizará la operación que está indicada por la instrucción.
4. Memory (MEM): No siempre existe esta etapa, solo cuando la instrucción indica transferencia con la memoria. Esta etapa escribe en la memoria la información.

5. Write (WB): Escribe los resultados de la ALU en el registro de destino o un registro recuperado de la memoria, si es que la etapa anterior ocurre.

Estos pasos se repiten una y otra vez en el procesador hasta que termine la ejecución. Cada fase tarda un ciclo de reloj, aunque algunas veces puede tardar dos ciclos o más en ejecutar una instrucción. El número de instrucciones es independiente de la cantidad de ciclos de reloj por realizar. La parte que se trabajó en este proyecto es el uso de la computación de propósito general, por lo que es necesario conocer el funcionamiento básico de la GPU. Por ende no es tan necesario comprender su funcionamiento orientado al procesamiento gráfico.

La velocidad de los procesadores aumentó con la frecuencia del reloj, a su vez el número de transistores por unidad de área aumentó con el tiempo. La frecuencia del reloj ha aumentado en casi cuatro órdenes de magnitudes entre el primer procesador 8088 Intel y los procesadores actuales, y el número de transistores también aumentó de 29,000 para Intel 8086 a aproximadamente 730 millones para un procesador Intel Core i7-9205. El aumento de la frecuencia de reloj tiene un efecto limitante importante, el calor disipado por los procesadores aumenta considerablemente por lo que aumenta el tamaño del sistema de enfriamiento limitando y puede causar un daño físico en el circuito, lo que se ha mantenido como una limitante en el diseño de circuitos integrados.

La Tabla 3.1 muestra información detallada sobre los componentes de cada tarjeta usada, haciendo énfasis en la cantidad de transistores existentes en cada tarjeta. Hay que aclarar que la GeForce 745m no es una versión anterior a la GeForce 1070, por lo que, aunque conocemos la Ley de Moore y no podemos ver una duplicación en el número de transistores, podemos ver un aumento considerable en la cantidad de transistores y una diferencia en el tiempo de ejecución.

Componentes	GeForce 1070	GeForce 745m
Arquitectura	Pascal	Kepler
Cuda cores	1920	384
Punto flotante	6.5 TFlops	642.8 GFlops
VRAM	8 Gb	2 Gb
Transistores	7.2 B	1270 M
Ranking de tarjetas	40	502

Tabla 3.1: Lista de los componentes más importantes involucrados en el poder de procesamiento de las GPU usadas en el proyecto.

3.1. Tarjetas gráficas

Existe una gran cantidad de tarjetas gráficas que están diseñadas para diferentes necesidades en específico. La tarjeta de video que se usó para esta investigación fue una NVIDIA Geforce GTX 1070 y una NVIDIA Geforce 745m. Actualmente, NVIDIA cuenta con un extenso catálogo de tarjetas gráficas definido por series y modelos. Definimos entonces tres tipos de tarjetas NVIDIA: Geforce, Quadro y Tesla.

Cada tarjeta está diseñada para necesidades específicas. NVIDIA Tesla fue orientada al cómputo científico o de propósito general. La NVIDIA Geforce se diseñó para los juegos de computadora y las labores que requieren rápido renderizado de polígonos como salida y visualización de recorridos en tercera dimensión. La NVIDIA Quadro se creó para el diseño, animación y el renderizado 3D, dándole una mayor prioridad en su rendimiento sobre Geforce. En términos más generales, la arquitectura de una tarjeta es diferente a la microarquitectura y algunas tienen nombres iguales, así por ejemplo, una tarjeta Geforce puede tener una microarquitectura Tesla o una tarjeta Quadro puede tener una microarquitectura Kepler, sin mencionar el catálogo de series como podrían ser Geforce RTX (o M, GT, GTX, etc.) o las Quadro series P, o la etiqueta "TI"para versiones mejoradas de una misma serie o

generación, solo por mencionar algunos diferenciadores de las tarjetas.

Existe una diferencia notable al usar una tarjeta para un trabajo que no está destinada. Por ejemplo, una NVIDIA Quadro renderizará una ciudad en 3D en 10 minutos, mientras que una tarjeta Geforce podría renderizarla hasta en media hora. Otro ejemplo es el apartado gráfico en un videojuego el cual rendirá mejor con una Geforce que una Quadro. Ahora bien, debemos tener en cuenta que las tarjetas que se comparan tienen que ser de la misma generación, ya que una tarjeta ajena a su propósito, de una generación mas reciente, podría ejecutar un mejor trabajo que una tarjeta dedicada.

3.2. Arquitecturas

La arquitectura de una computadora es la descripción del modelo conceptual del diseño de las partes con las que opera, la arquitectura también define el conjunto de instrucciones (ISA, *Instruction Set Architecture*) de cómo opera sobre estas, su direccionamiento y sus registros, por medio del procesador. En la actualidad, las computadoras se basan en una arquitectura de procesador x86 o x64, esto representa una arquitectura ISA. La forma en la que se almacenan los datos depende de la arquitectura que se está utilizando. Por ejemplo, un procesador de x64 almacena los datos en un espacio de 64 bits. Así, diferentes procesadores pueden tener diferentes arquitecturas para operar sobre los datos, por lo que es una de las causas principales de que se tengan problemas de compatibilidad en la ejecución de datos para diferentes arquitecturas como, por ejemplo, ejecutar un programa de una arquitectura x64 en una arquitectura x32. En 1966, Michael J. Flynn propuso la siguiente clasificación para los distintos tipos de arquitecturas:

- *Single Instruction Single Data* (SISD). En este tipo de arquitectura un solo procesador ejecuta instrucciones para las cuales almacena un único dato a la

vez que opera. Este modelo cuenta con registros de instrucciones, direcciones de memoria y datos de memoria.

- *Multiple Instruction Single Data* (MISD). Este tipo de arquitectura suele usarse para el cómputo paralelo. Usando un conjunto de instrucciones, diferentes unidades de procesamiento realizan diferentes operaciones y ejecutan diferentes instrucciones en el mismo conjunto de datos que se tenga. Suele usarse en computadoras para la detección de errores.

- *Single Instruction Multiple Data* (SIMD). Esta arquitectura tiene un conjunto de instrucciones con una unidad de control y muchas unidades de procesamiento. Ejecuta un conjunto de instrucciones secuenciales en el procesador, y ejecuta la misma operación en distintos datos.

- *Multiple Instruction Multiple Data* (MIMD). En esta arquitectura, el conjunto de instrucciones independientes del conjunto de datos que se tienen, es ejecutado por muchos procesadores. Suele usarse la memoria compartida o memoria distribuida para la ejecución de simulaciones y modelado, por ejemplo.

3.3. Microarquitecturas

La microarquitectura, a diferencia de la arquitectura, representa las conexiones entre buses [2] y componentes de una computadora por medio de diagramas de bloques. Las limitaciones en el avance del rendimiento de las computadoras es provocado principalmente por la fabricación de los circuitos integrados y la imposibilidad de paralelizar los procesos. Para combatir esta problemática, se desarrollaron los procesadores con múltiples núcleos que se dedicarían a paralelizar procesos. Las primeras GPU contenían una pequeña cantidad de núcleos que fue aumentando con el paso del tiempo, habiendo diferencias notables de procesamiento entre las que contenían mayores y menores núcleos. Esta diferencia de procesamiento de cómputo fue de-

[2] Los buses son el cableado físico mediante el cual viaja la información entre los componentes de la tarjeta en la que se encuentran.

finida por la microarquitectura de los procesadores, que es la manera en que las instrucciones son ejecutadas en el procesador, esto es, la conexión entre hilos, buses, unidades, bloques, entre otros, así como la cantidad y dirección de procesamiento. NVIDIA se basó en el tipo de arquitectura llamada SIMD (*Single Instruction, Multiple Data*), usando la representación *little endianness* .

3.3.1. Tesla

La microarquitectura Tesla se organiza en un clúster que contiene ocho procesadores que se clasifican como de alto nivel. Cada clúster está compuesto por una unidad de textura y dos multiprocesadores de transmisión (SM). Cada SM tiene ocho núcleos CUDA (procesadores de datos de CUDA[3]), dos unidades de función especial (estas son responsables de funciones que no pueden ser expresadas a través de expresiones polinomiales, cuadrados, raíces u operaciones trigonométricas, por dar ejemplos), una unidad de búsqueda de instrucción, una unidad de problema con memoria caché de instrucciones y 16 KB (Kilo bytes) de memoria compartida. La memoria compartida, a su vez, se divide en dieciséis bancos de palabras consecutivas de cuatro bytes, cuando cada banco es solicitado por distintos *threads*, un warp[4]. Cuando hay múltiples lecturas de *threads* por el mismo banco, un mecanismo se activa satisfaciendo todas las peticiones simultáneas de *threads*.

3.3.2. Fermi

La microarquitectura de Fermi trajo mejor organización en el multiprocesador de transmisión. Esta nueva arquitectura implementó el clúster de procesos gráficos (GPC), que remplazó al clúster de procesos (implementado en la arquitectura Tesla). Este nuevo clúster se compone por cuatro unidades de textura dedicada, reempla-

[3]Del inglés *Compute Unified Device Architecture*, es la plataforma de computo en paralelo usada en este proyecto. Esta plataforma es descrita en la sección 3.4

[4]Un *warp* es la unidad de procesamiento de CUDA, está definido como un conjunto de 32 hilos de ejecución

zando a las que tenía Tesla e incrementando el número de SM a cuatro en total. Los nuevos SM contenían 32 núcleos CUDA y un nuevo caché configurable, dando más libertad al programador. La memoria compartida total se renovó de su antecesor cambiando la capacidad total de su memoria a 48 KB. La capacidad de las unidades de función especial fue duplicada y cada hilo se independizó cada vez más de los procesos de los demás. El tiempo de carga de los procesos es dividido en grupos de 16 núcleos de CUDA y las instrucciones son distribuidas por dos *warp* permitiendo que sean procesados y ejecutados al mismo tiempo.

3.3.3. Kepler

Esta microarquitectura mejoró lo implementado en Fermi, disminuyendo el poder de consumo total, es decir, usa menos energía para sus procesos. Esta arquitectura de alto nivel mantuvo el mismo sistema de clúster implementado en Fermi, el GPC, pero el SM pasó a ser nombrado SMX. Cada SMX contiene ahora 192 núcleos CUDA, 32 unidades de carga y almacenado y 32 unidades de función especial. Logrando, también, doblar el número de planificadores warp y así incrementar el número de despachadores de instrucción a dos, ayudando a los warps a ejecutar instrucciones independientes en tiempo menor en comparación con arquitecturas pasadas.

3.3.4. Maxwell

La microarquitectura Maxwell mejoró a su antecesora, Kepler, dando una mejor eficacia de poder y mejora de rendimiento. Los SMX, ahora nombrados como SMM, disminuyeron en cantidad de núcleos, pasando a ser 128, en comparación de los 192 que tenía Kepler. Maxwell mantuvo la cantidad de unidades de ejecución especial. La organización de los SM pasó a cuatro grupos con 32 núcleos CUDA, 8 unidades de carga y almacenado, 8 unidades de función especial. La memoria compartida se

convirtió en una unidad dedicada con mayor capacidad a 96 KB, superando a más del doble la capacidad de las arquitecturas Fermi y Kepler.

3.4. Orientación a computación de propósito general en unidades de procesamiento de gráficos (GPGPU)

Con el avance de la tecnología las GPU han logrado tener una gran potencia de cómputo gráfico. En los últimos años se ha empezado a utilizar el poder de cómputo de las GPU para realizar tareas no gráficas. Por ejemplo, se ha utilizado en aplicaciones de bioinformática, simulaciones y análisis numérico. La razón es que las GPU tienen tasas de operaciones de punto flotante mucho mayores, incluso que las CPU de múltiples núcleos, se debe a que las GPU están especializadas para cálculos intensivos altamente paralelos y están diseñadas con muchos más transistores asignados al procesamiento de datos en lugar de al control de flujo o almacenamiento de datos en caché [Oancea et al., 2014]. Para aprovechar al máximo los recursos que ofrecen, es necesario diseñar la ejecución de código en paralelo.

Antes del surgimiento de esta nueva perspectiva de uso de GPU de propósito general, resultaba difícil la manipulación tanto de los procesos de ejecución como de los bloques de memoria para reducir los tiempos de cómputo. De esta forma surgió la plataforma CUDA (*Compute Unified Device Architecture*). Esta plataforma está orientada a la computación en paralelo en la GPU y facilita la computación de aplicaciones de propósito general. Esta fue la plataforma que se usó para lograr paralelizar los procesos en este proyecto.

3.4.1. CUDA

CUDA es una arquitectura de procesamiento en paralelo para el alto desempeño de cómputo en GPU creada por NVIDIA que surge a partir de la necesidad de descentralizar los procesos de la CPU usando la GPU como co-procesamiento de información central. CUDA está basada en el lenguaje de programación C/C++.

El uso de CUDA, como programador, se puede implementar con diferentes lenguajes de alto nivel, tal como lo es C, C++, Fortran, Python, aprovechando drásticamente el poder de las GPU y reduciendo el tiempo computacional. Una característica única de CUDA, es que los núcleos tienen registro y memoria compartida integrada en el chip. Esto permite compartir información sin que pase por el bus de memoria del sistema agilizando el proceso. Además, cada núcleo puede tener las mismas y diferentes instrucciones usando los mismos recursos. [Storti and Yurtoglu, 2016]

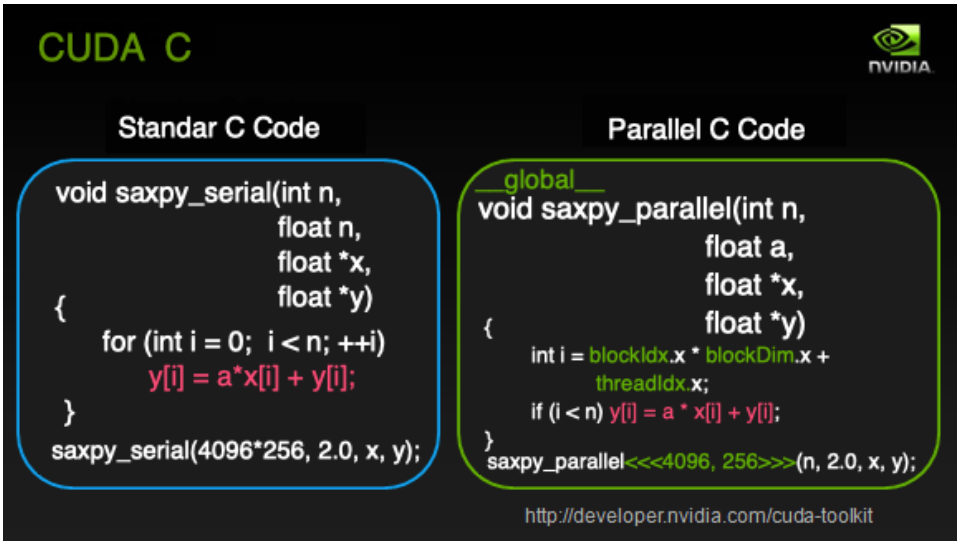

Figura 3.2: La llamada a la función CUDA en C es mediante el nombre de la función seguido por el número de bloques y el número de hilos que se van a ejecutar en la función, y las entradas de los argumentos. Imagen obtenida de CUDA Toolkit Documentation v10.1.105

3.4.2. Arquitectura CUDA

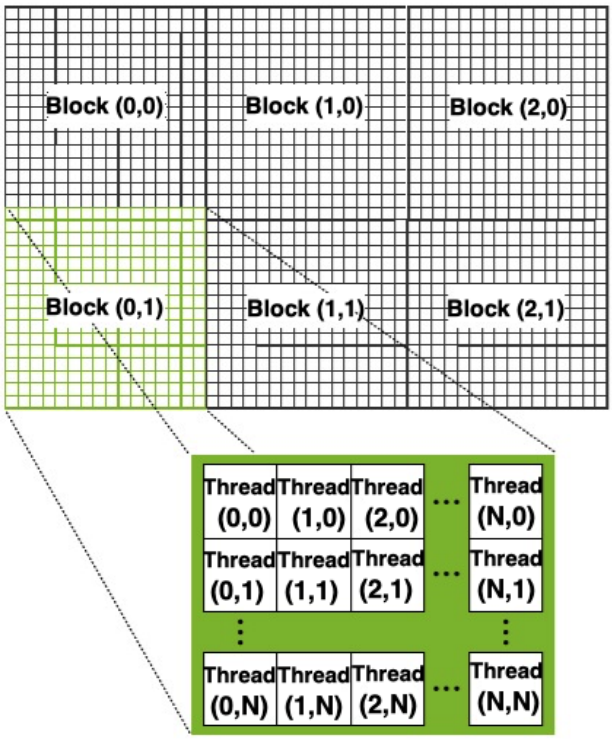

Figura 3.3: Un malla (grid) está formado por bloques (blocks). Un bloque está formado por hilos (threads). Un hilo es el script de ejecución. Imagen obtenida de la página oficial de NVIDIA (www.nvidia.com)

La arquitectura de CUDA se compone por una matriz, donde cada elemento o unidad de proceso, que son los *Streaming Multiprocessors* (SM), se conectan con todos los demás SM en una zona de memoria. A su vez, un SM está compuesto por núcleos llamados CUDA core o *Streaming Processors* (SP), que ejecutan las instrucciones. Existe un mismo número de SP para cada SM, permitiendo, así, un diseño de mayor flexibilidad a la hora de programar sobre los núcleos en la GPU.

Al script (en cualquier lenguaje de alto nivel que permita CUDA) encargado de manipular los núcleos se le llama kernel y puede llegar a ser desde una simple línea

de código o una función hasta un complejo programa. Este kernel se ejecutará en un conjunto de hilos, que a su vez se organizan en bloques y al mismo tiempo, los bloques se organizan formando una malla. Por conveniencia y comodidad, tanto la malla como los bloques pueden tener una arquitectura de hasta tres dimensiones. Es decir, su representación puede verse como una matriz de una, dos o tres dimensiones [Sanders and Kandrot, 2010]. Los hilos del bloque, al igual que los bloques en la malla, pueden ejecutarse de manera independiente o ejecutarse en sincronía. Estos se ejecutan al mismo tiempo en un conjunto de 32 hilos llamado *warp*, al cual se le conoce como la unidad de ejecución en paralelo de CUDA debido a que todos los hilos se ejecutan en paralelo y desde la misma instrucción. Los bloques de la malla pueden ejecutarse independientemente de los otros y de igual manera, pueden ejecutarse en paralelo. Al ejecutar un bloque, este se divide en *warps* para después seleccionar otro *warp* y ejecutar las instrucciones de cada hilo que lo conforma. Dentro del kernel de ejecución se especifica la cantidad de hilos que se requiere ejecutar dentro de cada bloque, siendo la misma cantidad para cada bloque, y también se identifica la cantidad de bloques por malla. Tanto para cada hilo, como para cada bloque, se asignará un identificador que especifica dentro de qué bloque está (en el caso de los hilos) y su posición, y para cada bloque su posición en la malla. Para computar la llamada del kernel, la función CUDA está definida de la siguiente manera:

$$myKernel <<< blocks, threads >>> (arg_1, arg_2, ..., arg_n);$$

En la llamada a una función CUDA se especifica el nombre de la función que se ejecutará- (mykernel), dentro de los paréntesis " <<< " y " >>> " se indica el número de bloques e hilos que ejecutará la función, seguido por los argumentos de entrada para la función.

Con esta arquitectura podemos ejecutar diferentes hilos para diferentes tareas lo que libera recursos en el direccionamiento de memoria. Cada hilo se encuentra en una zona privada de la memoria local y cada bloque, en una zona compartida a la que cada hilo del bloque puede acceder. Al mismo tiempo, cada hilo puede acceder a un espacio de la memoria global. CUDA asume que, tanto el host (CPU), como

el device (GPU) tienen espacios separados en memoria, pudiendo acceder de forma conjunta solamente a la memoria global.

3.4.3. Acceso a la memoria

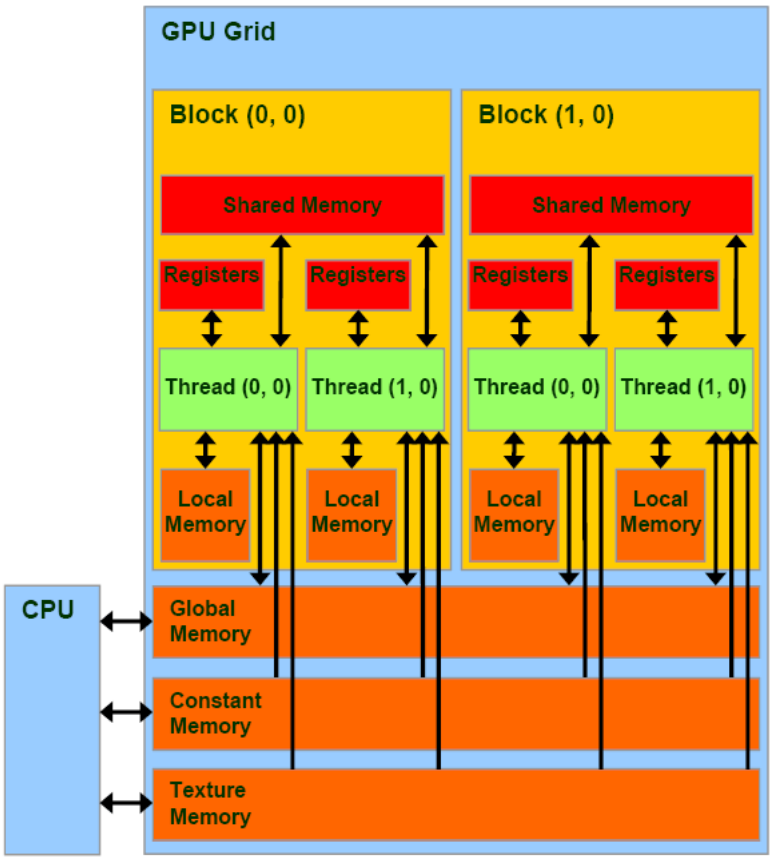

Figura 3.4: Acceso a los bloques de memoria desde el CPU y al interior del GPU [NVIDIA Corporation, 2018].

Para que el kernel pueda ser ejecutado necesita acceder a la memoria. Para ello es necesario reservar memoria para su ejecución y, posteriormente, liberarla al terminar el proceso, todo mediante líneas descritas en el kernel. El host, que representa al CPU, puede acceder a la memoria global y a la memoria constante del device,

que representa al GPU, para así poder transferir la información a la GPU. Por otro lado, los hilos pertenecientes al device pueden acceder a la memoria compartida. La memoria funciona para devolver la información o dar una respuesta de los datos al host. Esta memoria existe hasta que se "mata" el proceso o que se libera la información. La memoria local es la que está reservada para la información de cada hilo. Para la memoria global, el acceso y la información es visible para todos los hilos pertenecientes a cualquier bloque. Es decir, cualquier hilo de la malla puede leer y escribir en la memoria global. La memoria compartida funciona de manera que separa el acceso por cada bloque de la malla, los hilos pertenecientes a un bloque pueden leer y escribir en la memoria compartida que pertenece a ese bloque; pero un hilo de un bloque diferente no puede leer ni escribir en la memoria compartida de otro bloque. La memoria constante tiene un funcionamiento parecido a la memoria global. Esta memoria se usa para mantener los datos, que no cambian, durante la ejecución del kernel, sin poder modificar esta información. La organización de los bloques de memoria se muestran en la Figura 3.4.

En resumen, podemos decir que cada hilo tiene acceso a espacios de la memoria durante el proceso. Asimismo, cada hilo tiene su propia memoria local, cada bloque tiene su propia memoria compartida (disponible para cada hilo del bloque) y todos los hilos tienen acceso a la memoria global. Cuando se asigna espacio en la memoria global desde el kernel, los datos necesarios se transfieren entre las memorias desde el procesador, esto mediante una transferencia asíncrona. Lo anterior se hace indicando cuatro parámetros, el primero apunta hacia la dirección destino de los datos, es decir, a donde se tienen que copiar los datos. El segundo indica cuáles datos se copiarán. El tercero el número de bytes que se van a copiar y el cuarto indica el tipo de memoria en la copia, esto es, de la memoria principal a la misma memoria del procesador, de la memoria del procesador principal a la memoria del dispositivo destino, de la memoria del dispositivo a la memoria principal o de la memoria del dispositivo a la misma memoria, es decir, hace una transferencia de datos entre el mismo dispositivo, no entre dispositivos.

CAPÍTULO 4

SISMICIDAD Y TECTÓNICA DE MÉXICO

Contenido

A continuación daremos una breve introducción a la sismología y al problema que pretendemos resolver a través del uso de una GPU. La sismología es la ciencia encargada del estudio de los terremotos, sus causas y sus consecuencias. En México, la sismología ha tomado particular importancia debido al ambiente tectónico en el que se encuentra el país. Se genera un gran número de sismos cada año, los cuales pueden tener afectaciones importantes sobre la población y la infraestructura. En este sentido, el avance de la herramientas computacionales, tanto de *software* como de *hardware*, han sido de suma importancia para el procesamiento de un volumen cada vez mayor de información y para el desarrollo de modelos numéricos cada vez más realistas que nos permitan conocer mejor cómo es que ocurren estos fenómenos naturales. En este capítulo, definiremos algunos de los conceptos básicos de la sismología y haremos una revisión de la tectónica de placas. Además de introducir al lector en el análisis de sismos repetitivos, subrayando la importancia de su estudio y el por qué es necesario el uso de procesadores gráficos para su detección.

4.1. Conceptos básicos

4.1.1. Tipos de onda

Los terremotos generan diversas ondas sísmicas que se pueden catalogar de acuerdo con el movimiento que producen, lo que determina tanto su amplitud como la velocidad con la que se propagan. De forma general, las ondas sísmicas se pueden dividir en dos tipos principales: (a) las ondas de cuerpo y (b) la ondas superficiales.

Se conoce como **ondas de cuerpos** a las ondas sísmicas cuya trayectoria de propagación se da principalmente en el interior de la Tierra. Estas, a su vez, se pueden clasificar en ondas primarias (ondas P) y en ondas secundarias (ondas S). Las **ondas P**, también conocidas como ondas acústicas, causan la convergente y desconvergente del medio en la misma dirección de la propagación de igual forma que las ondas sonoras que viajan por el aire. Son las ondas sísmicas que viajan con mayor velocidad y son las primeras en observarse en los registros sísmicos, aunque típicamente son las ondas de menor amplitud con respecto a los otros tipos de ondas. Posterior a la llegada de las ondas P se observa el arribo de las **ondas S**. Las ondas S, a diferencia de las ondas P, generan un movimiento perpendicular a la trayectoria de propagación y tienen una amplitud mayor que las P. Este tipo de ondas únicamente pueden propagarse en medios sólidos, es decir, no pueden propagarse en líquidos o gases. La ausencia de ondas S permitió determinar que que el interior de la Tierra está formado de un núcleo líquido de níquel y hierro.

Por otra parte, las **ondas superficiales** viajan principalmente cercanas a la superficie terrestre. Esto ocasiona que la energía producida por el sismo viaje a través de un volumen menor y consecuente la amplitud de estas ondas sea mucho mayor que las de cuerpo. Estas ondas son las principales responsables de los daños causados por los sismos. La velocidad de propagación de estas ondas es menor que las ondas de cuerpo y se pueden clasificar en dos tipo: ondas Rayleigh y ondas Love.

Las **ondas Rayleigh** producen un movimiento circular en sentido contrario a la propagación, parecido a las olas en el mar. Son ondas de baja frecuencia cuyo periodo está en el rango de los 10-100 s. Por último, las **ondas Love** son otro tipo de onda de superficie que a diferencia de las Rayleigh, producen un movimiento del subsuelo de tipo horizontal perpendicular a la dirección de propagación. Al igual que las ondas Rayleigh pueden causar daños serios a viviendas y edificios.

Es importante recalcar que la mayor parte de los sismos analizados en el presente proyecto no producen, o producen en baja medida ondas superficiales debido a su baja magnitud y cercanía con la estación. En la Figura 4.1 se ilustra como ocurre la propagación de estos tipos de onda, en la Figura 4.2 se muestra como son registradas estos tipos de ondas por una estación sísmica.

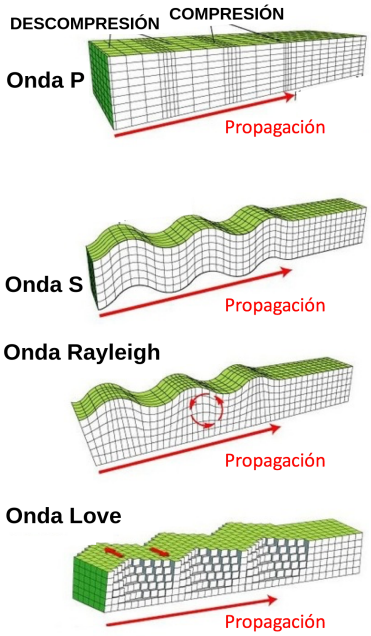

Figura 4.1: Desplazamiento y trayectoria de propagación para los cuatro tipos principales de ondas sísmicas. Modificado de https://www.sciencelearn.org.nz/resources/340-seismic-waves. Accesado: Septiembre 2019.

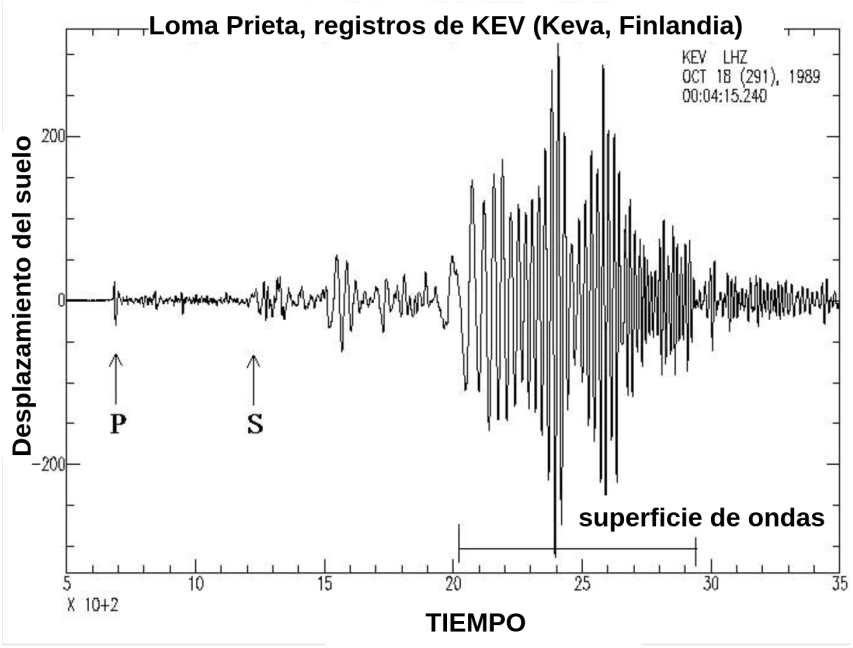

Figura 4.2: Componente vertical del sismo de 1989 en Loma Prieta, California registrado en Finlandia. Fuente: www.gns.cri.nz/outreach/qt/quaketrackers/curr/seismic_waves.htm

4.1.2. Características de un sismo

Los sismos pueden caracterizarse para su estudio de acuerdo con diversos parámetros. El primero de ellos es su localización, la cual está dada por la longitud y latitud del punto en el que se inicia la ruptura del mismo. Al punto en superficie directamente por encima del sitio de inicio de la ruptura se le conoce como **epicentro**, mientras que al lugar exacto por debajo de la superficie terrestre se le conoce como **hipocentro**. En la Figura 4.3 se ilustra la diferencia entre epicentro e hipocentro. Por otra parte, la magnitud de un sismo (medida en Newtons Metro (Nm)) se puede estimar a partir del momento sísmico, M_0, que se calcula a partir de conocer el área de ruptura A, el desplazamiento promedio d y el coeficiente de fricción μ. De acuerdo a la siguiente ecuación:

$$M_0 = Ad\mu\, Nm. \tag{4.1}$$

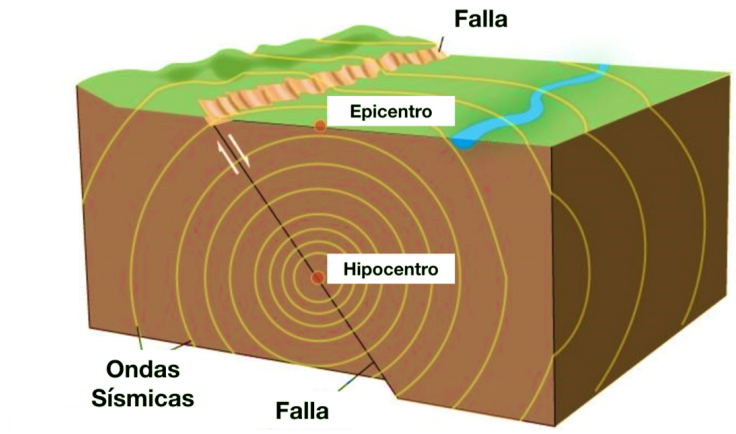

Figura 4.3: Elementos básicos que describen la localización de un sismo con respecto al plano de falla.

El área de ruptura de los sismos puede variar de unos cuantos metros cuadrados para sismos de magnitudes pequeñas (M2-3), hasta varios cientos de miles de kilómetros cuadrados para grandes sismos $> M8$. Por lo que el momento sísmico, M_0, es un número que puede abarcar varios ordenes de magnitud. Por este motivo, el momento sísmico suele expresarse como una cantidad adimensional conocida como magnitud de momento, que es reportada por diversos servicios sismológicos del mundo y por lo medios de comunicación (radio, televisión). La magnitud de momento se obtiene a partir de la siguiente relación empírica [Hanks and Kanamori, 1979],

$$M_w = \frac{3}{2}(\log_{10} M_0 - 9.1). \tag{4.2}$$

Esta relación da como resultado un número típicamente entre 1-10, que está relacionado con el energía producida por un sismo (aunque puede tomar valores negativos, para sismos muy pequeños que únicamente pueden ser detectados por sensores a unos cuantos cientos de metros de la falla que los origina). Hasta la fecha, el sismo más grande registrado por el hombre ocurrió el 22 de mayo de 1960, en la costa de Chile con una $M_w = 9.5$. El sismo del 19 de septiembre de 1985, ocurrido en las costas de Michoacán y que causó grandes daños en la Ciudad de México tuvo una magnitud de $M_w = 8.1$, y el sismo con epicentro en Puebla, que dañó gravemente la

Ciudad de México nuevamente el 19 de septiembre pero del 2017, tuvo una magnitud de $M_w = 7.2$.

4.2. Tipos de fallas

Los sismos ocurren en fracturas en el interior de la Tierra que pueden tener desde unos cuantos metros de longitud hasta cientos de kilómetros. Estas fallas son el resultado del movimiento de las placas tectónicas y del efecto de la gravedad. Las fallas se clasifican de acuerdo al movimiento que tiene cada uno de los bloques que forman la falla con respecto al otro. Las **fallas normales** ocurren cuando uno de los bloques que forma la falla "cae" con respecto a la otra. Esto ocasiona que uno de los bloques se desplace hacia abajo con respecto al otro. Este tipo de fallas pueden ocurrir por efectos de la gravedad o en ambientes tectónicos divergentes, por ejemplo en zonas donde dos placas **divergen** entre sí, como ocurre en el Golfo de California. Por otra parte, las **fallas inversas** ocurren cuando dos bloques **convergen** entre sí. Este es el tipo de falla dominante a lo largo de la costa del Pacífico. El tercer tipo de falla, son las **fallas transformantes**. Estas fallas se originan cuando los bloques que forman las fallas se deslizan horizontalmente entre sí. La falla más conocida de este tipo, es la llamada **Falla de San Andrés**, la cual atraviesa casi en su totalidad el estado de California, en los Estados Unidos. La Figura **??** ilustra estos tres tipos de falla.

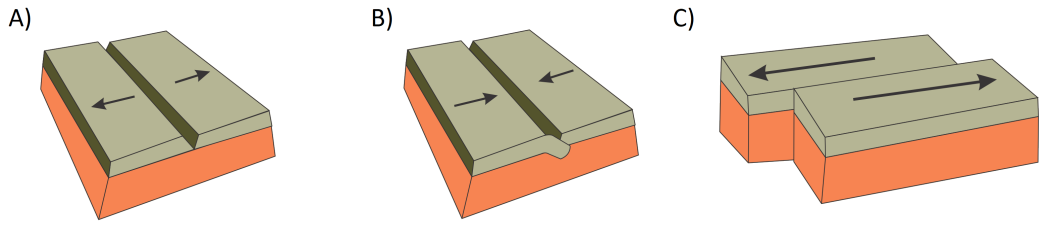

Figura 4.4: A) Falla normal con movimiento divergente, B) falla inversa o convergente y (C) falla transformante. Estos tipos de falla, están asociados a los diferentes límites de placas.

4.3. Sismicidad de México

La sismicidad de México se debe principalmente a la colisión de cinco placas tectónicas que rodean al país. Estas placas son:

- La placa Norteamericana. Se extiende en Canadá, Estados Unidos, el occidente del Océano Atlántico Norte, y parte del Océano Glacial Ártico.
- La placa del Caribe. Forma parte de América Central y el mar Caribe.
- La placa del Pacífico. Cubre gran parte del océano Pacífico.
- La placa de Cocos. Causa la mayor parte de la sismicidad y que se subduce[1] de forma casi perpendicular en los estados de Chiapas, Oaxaca, Michoacán, Jalisco y Guerrero.
- La placa de Rivera. Es una microplaca localizada al norte de la placa de Cocos, colindante principalmente con el estado de Colima. Aunque la sismicidad anual de esta placa es una de las más bajas, produjo el sismo más grande registrado instrumentalmente en México, el 3 de junio de 1932 que ocasionó un devastador tsunami que afecto principalmente la ciudad de Manzanillo (el sismo ocurrido el 7 de septiembre del 2017, tuvo una magnitud muy similar).

El movimiento producido entre cada una de las placas es sumamente variado. La placa del Caribe y la placa Norteamericana tienen un movimiento transformante entre sí, que forma la falla de Montagua-Polochic. La placa del Pacífico y la Norteamericana también tienen un movimiento transformante, pero en el noroeste de estas dos placas el movimiento es divergente formando así el actual Golfo de California. En las placas del Pacífico y la de Cocos y Rivera existe un movimiento divergente que da origen a una cadena montañosa volcánica submarina que delimita el borde de las placas. Cocos y Rivera producen un movimiento convergente con la placa Norteamericana. Estos movimientos producen la mayor parte de la sismicidad, cuando

[1] Se entiende por subducir, cuando una placa téctonica se hunde debajo de otra.

la roca que compone la corteza terrestre se rompe por una fuerza que actúa sobre ésta. Generalmente el choque de las placas tectónicas produce fricción que acumula fuerza durante el movimiento hasta que la fuerza sea mayor a la fricción provocando una liberación de energía en diferentes formas, como calor, deformación de rocas y energía sísmica, que se libera a la superficie. El rompimiento de la corteza determina el tamaño del sismo, esto es, a mayor rompimiento se produce mayor energía sísmica. Por lo tanto, los sismos pueden originarse de diferentes formas:

- **Sismos de origen tectónico**. Son los que se producen por la interacción de las placas tectónicas, estas tienen diferentes tipos de movimientos que son los causantes de estos sismos. Podemos clasificar este tipo de sismos dependiendo del lugar donde ocurre la ruptura en:
  - **Interplaca**. Son originados en los límites de las placas tectónicas. En general, estos sismos son los que representan un mayor riesgo debido a su frecuencia y a que pueden alcanzar magnitudes superiores a los $\geq M8$.
  - **Intraplaca**. Son originados al interior de la placa tectónica. A pesar de que este tipo de sismos son menos frecuentes. Los sismos del 7 de septiembre, y 19 de septiembre del 2017, en Oaxaca y Puebla, caen en esta categoría. La ruptura ocasionada por estos sismos, es producida por una ruptura en la placa misma y no por el deslizamiento producido por el choque con otra placa.
- **Sismos de origen volcánico**. Este tipo de sismos es provocado por el movimiento ascendente de los fluidos de los volcanes, y se perciben durante los días previos a las erupciones volcánicas. A diferencia de los sismos tectónicos, estos tienen formas de onda (*sismogramas*) con contenido espectral muy diferente. Además, generalmente suceden en forma de **enjambres**, es decir, en grupos de sismos que ocurren con mucha frecuencia en un periodo de unos cuantos días y que posteriormente cesan su actividad.

Existen zonas que tienen ciertas características que provocan que un sismo ocurra con más facilidad. En México, los estados que se encuentran cerca de los límites de

placas son los que tienen más riesgo de alta sismicidad. Existen zonas, que aunque se encuentran lejos de los límites de placas, pueden tener fallas activas. Este es el caso de los sismos reportados por el SSN (Servicio Meteorológico Nacional), en el estado de Nuevo León. La Figura 4.5 muestra los principales sismos ocurridos en los últimos 100 años. Los sismos alejados de la costa, pueden tener efectos devastadores sobre las ciudades debido a la cercanía y a la alta densidad demográfica del centro de México.

4.4. Sismos repetitivos

Los sismos repetitivos son un tipo particular en el que nos enfocaremos para desarrollar este proyecto. Esta variedad de sismos se caracterizan por tener formas de onda casi idénticas y tasas de recurrencia (tiempo entre sismos consecutivos) relativamente regulares. Este tipo de sismos se reportaron por primera vez en California, a lo largo de la falla de San Andrés y a lo largo de la falla de Calaveras en 1994 [Vidale et al., 1994]. Algunas de las similitudes que comparten estos sismos son la magnitud, la localización del hipocentro y el mecanismo focal que los produce. Es decir, si es un sismo de falla, normal inversa o transformante, ver Figura 4.4. Esto genera que los sismogramas registrados sean casi idénticos. Los sismos repetitivos se presentan cuando al menos dos o más sismos ocurren con las características antes mencionadas. Las definiciones se basan en la relación de superposición de áreas de origen estimadas y el grado de similitud en la forma de onda. La determinación del hipocentro que puede confirmar la colocación real de los terremotos, representa una manera de identificar sismos repetitivos [Uchida and Bürgmann, 2019].

En la Figura 4.6, se muestra de forma esquemática el origen y cálculo de la tasa de deslizamientos a lo largo de una zona de subducción. A medida que la placa oceánica se desliza por debajo de la corteza continental genera sismos como consecuencia de las asperezas que existen en el contacto entre las placas. Algunas de estas

asperezas se rompen en forma de grandes sismos de subducción, mientras que en otros casos la placa es capaz de deslizarse de manera asísmica, es decir, generando grandes desplazamientos de masas detectados por las estaciones de GPS en tierra, pero sin generar ondas sísmicas detectables por estaciones sísmicas.

Entonces, los sismos repetitivos permiten monitorear cómo es que ocurren estos desplazamientos entre las placas tectónicas a profundidades en las que no es posible una instrumentación directa. [Nadeau and Johnson, 1998] establecieron la siguiente relación empírica que permite conocer la tasa de deslizamiento en la interfaz de las fallas a partir de la magnitud de momento. La siguiente ecuación permite conocer el deslizamiento producido por un sismo repetitivo:

$$log(d) = -2.36 + 0.17log(M_0). \tag{4.3}$$

Donde d es el desplazamiento en cm, y M_0 es el momento sísmico en $dinas \cdot cm^2$.

La Figura 4.6 muestra de manera esquemática el procedimiento utilizado para el cálculo del desplazamiento. En el panel A se indica como pequeñas asperezas rompen de manera sucesiva a medida que acumulan y liberan esfuerzos, producto del movimiento entre las placas tectónicas. Si, efectivamente se trata de un sismo producido por la misma aspereza se espera que los sismogramas registrados por una misma estación en dos instantes diferentes de tiempo sean casi idénticas. La similitud de dos señales se puede determinar de forma matemática a través del coeficiente de correlación y la coherencia espectral, los cuales se describen en el Capítulo 5. Posteriormente, se analizan en una área determinada A, todas aquellas secuencias de sismos repetitivos cercanos entre sí. Una vez identificadas las secuencias, se procede al cálculo del deslizamiento producido por los diferentes miembros de un mismo grupo de sismos repetitivos. Para esto se aplica la Ecuación 4.3, a fin de estimar el desplazamiento de la falla debido a la ocurrencia del sismo calculada. A continuación, se promedia la suma acumulativa de todos los desplazamientos producidos por un grupo de sismos repetitivos detectados en el área A. Finalmente, se obtiene la historia de deslizamiento donde se observan los periodos de tiempos en los que la

[2]La dina es una unidad de medida de fuerza, aplicada a la masa de un gramo.

falla tuvo mayor actividad. Para este trabajo, se clasificaron como sismos repetitivos todos aquellos pares de sismos con coeficiente de correlación y coherencia espectral superior al 95 %. Diversos autores, han utilizado diferentes umbrales de detección para la clasificación de sismos repetitivos. [Uchida, 2019] hace una revisión bibliográfica de los diversos métodos utilizados y de los umbrales utilizados en diversas regiones del mundo.

Tomando en cuenta que los sismos grandes ocurren rara vez en comparación con los sismos pequeños, podemos decir que los datos que se tienen sobre sismos grandes no son suficientes y no existe una manera de demostrar que la distribución escalada de espacio-tiempo-magnitud de los sismos pequeños sea diferente de la de los sismos grandes [Zechar and Nadeau, 2012]. Para la correcta identificación de los sismos repetitivos mediante la correlación cruzada es necesario tener ventanas de tiempo lo suficiente extensas, altas frecuencias y un umbral de coherencia y de correlación alto. Para la determinación de estos sismos, es necesario tener un criterio alto ya que en muchos estudios con un criterio poco estricto se han determinado falsos positivos mientras que un criterio muy estricto puede ocasionar que se omita la detección.

A lo largo de las zonas de subducción, se ha observado la presencia de sismos repetitivos en regiones de relativamente alta sismicidad con coeficientes de acoplamiento variable [3]. En la Figura 4.7, se muestra el ejemplo de una secuencia de 7 sismos repetitivos ocurridos cerca de Pinotepa Nacional, Oaxaca entre 2008 y 2014. La cual fue confirmada por dos estaciones sísmicas del SSN (PNIG y TLIG). Del lado izquierdo, se muestran las formas de onda de los eventos, los cuales ocurrieron con mucho mayor frecuencia después del sismo M 7.4 del 20 de marzo del 2012 en Ometepec, Oaxaca. En la columna derecha, se muestra la localización reportada en el catálogo del SSN para estos eventos.

[3] El coeficiente de acoplamiento mide la relación entre la tasa de deslizamiento sísmico y la tasa de deslizamiento a largo plazo de la placa que subduce. El cual estima la capacidad que tienen las placas de resistir la acumulación de esfuerzos y deformaciones.

En este mapa, se observa que aunque las localizaciones del SSN son muy similares, no pueden ser utilizadas como criterio para la detección de sismos repetitivos debido a que estos sismos ocurren mar adentro donde no existe cobertura de estaciones y por lo tanto incrementa el error en la localizaciones. Sin embargo, el análisis tanto de los coeficientes de correlación como de la coherencia espectral de las señales permite corroborar que estos sismos muy probablemente son generados por una aspereza en la interfaz de la placa que incrementó su tasa de deslizamiento como consecuencia del deslizamiento producido por factores tanto sísmicos, como por asísmicos.

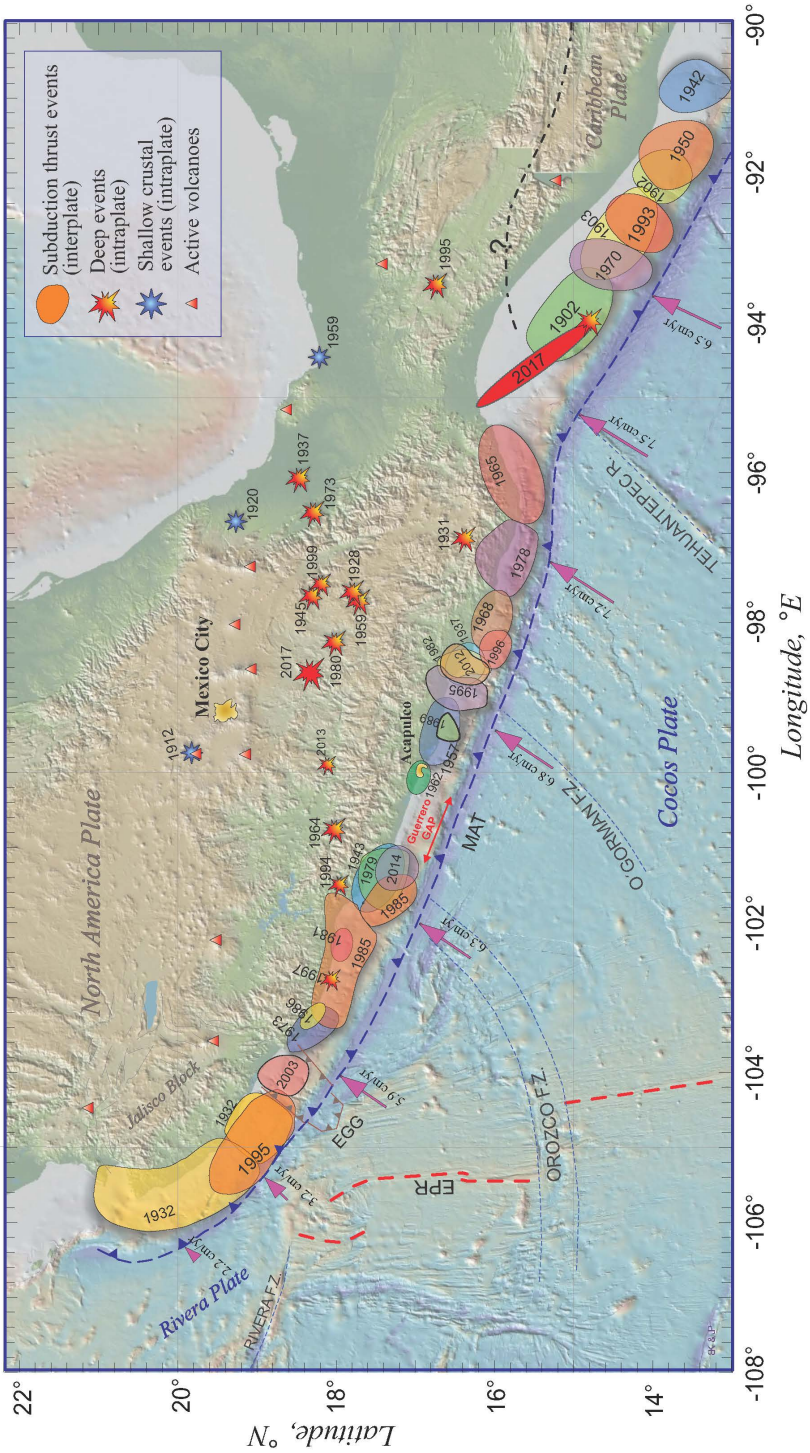

Figura 4.5: Zonas de ruptura de los sismos más importantes ocurridos en México en lo últimos 100 años. La mayor parte de los grandes sismos de México ocurren en la costa del Pacífico. URL: http://usuarios.geofisica.unam.mx/vladimir/sismos/100a%F1os.html . Recuperado: Octubre 2019.

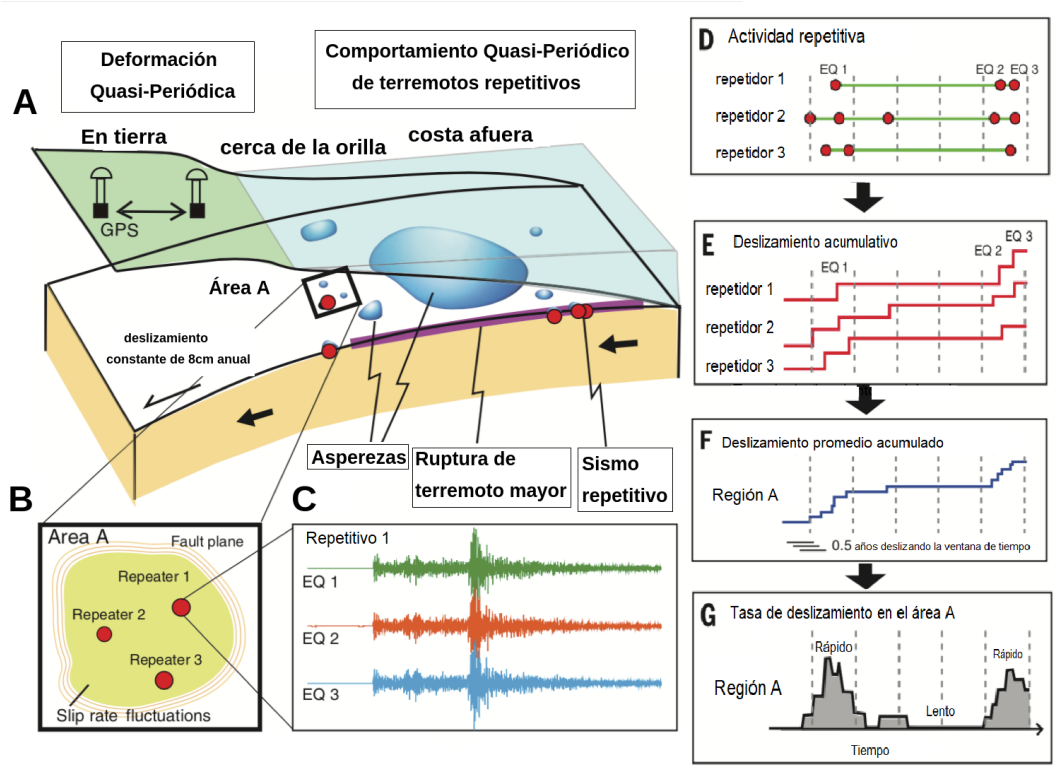

Figura 4.6: Cálculo de las tasas de deslizamiento a partir de sismos repetitivos. a) Los sismos repetitivos ocurren en la interfaz entre las placas a causa de asperezas de estas; b) y c) ejemplo de las formas de onda generadas por sismos repetitivos en un mismo plano de falla; d) detección temporal de los sismos repetitivos para el cálculo de las tasas de recurrencia; e) cálculo de desplazamiento de forma individual; f) desplazamiento promedio en el plano de falla; g) historia de deslizamiento. Modificado de [Uchida and Bürgmann, 2019]

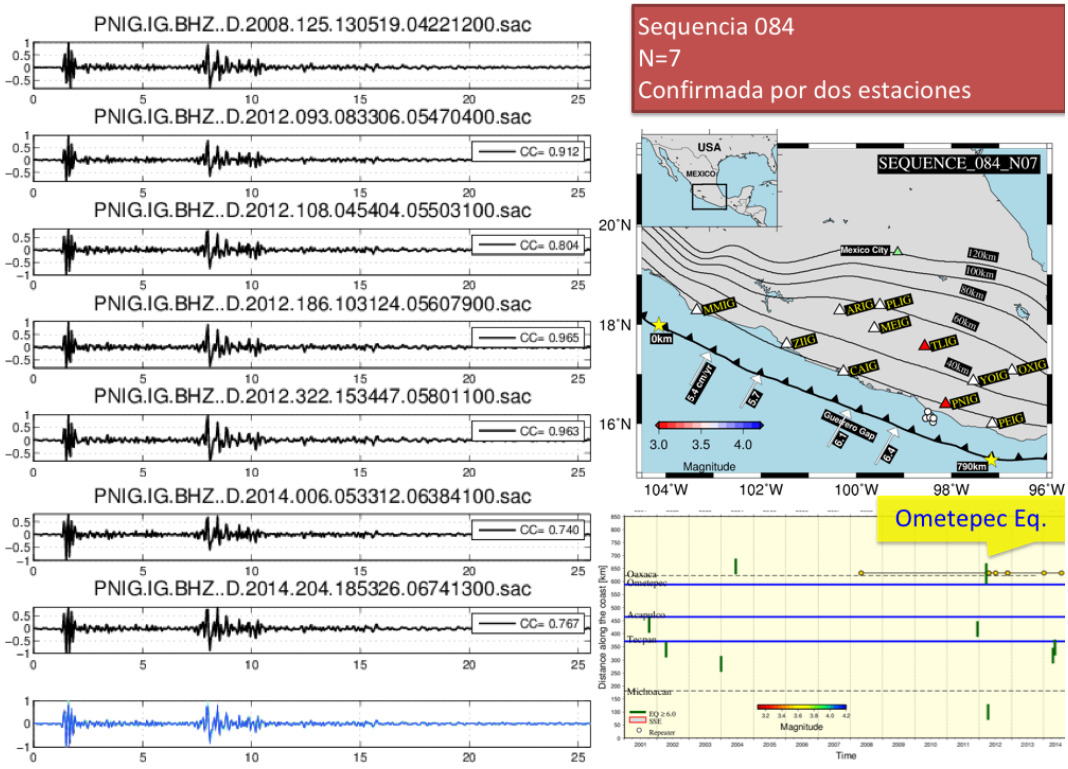

Figura 4.7: Ejemplo de secuencia de sismos repetitivos detectados en México. El mapa indica la localización de los sismos de acuerdo al Servicio Sismológico Nacional.

CAPÍTULO 5

PROCESAMIENTO Y ANÁLISIS DE SEÑALES

Contenido

El procesamiento digital de señales, es una de las ramas de la ingeniería que más se han desarrollado en los últimos años y es la base del presente proyecto. El principal avance surge a partir del cambio en la forma de almacenamiento de los datos de medios analógicos a digitales. Los sistemas digitales, a diferencia de los sistemas electrónicos analógicos, permiten una mayor flexibilidad en el almacenamiento

y procesado de los datos. En este capítulo examinaremos algunos de los conceptos básicos de procesamiento de señales, desde la adquisición y acondicionamiento de la señal hasta el procesamiento y filtrado de esta. En particular, pondremos énfasis en la Transformada de Fourier, que es una herramienta matemática que permite descomponer una señal en una sumatoria ponderada de senos y cosenos. Esta transformación tiene importantes propiedades matemáticas que permite analizar una señal de manera más eficiente mediante el análisis de sus componentes espectrales. Estas propiedades son de suma importancia para el desarrollo de nuestro objetivo principal que es la detección de sismos repetitivos.

5.1. Clasificación de señales

Las señales consisten en un conjunto de datos numéricos que representa las variaciones del estado de un fenómeno físico de un sistema real o simulado. Por lo general, las señales se expresan mediante funciones de una o varias variables independientes o co-dependientes ya sea en el dominio del tiempo o del espacio. En una primera clasificación, las señales se pueden catalogar como señales continuas o discretas, de acuerdo a la naturaleza del muestreo.

Las señales registradas en tiempo continuo, también conocidas como analógicas, son aquellas cuya frecuencia de muestreo es infinita, es decir, existe un valor de la señal para cualquier instante de tiempo o punto en el espacio. Por otro lado, las señales discretas, están definidas para un conjunto finito de valores. Existen medios para poder registrar señales analógicas como lo son las cintas magnéticas y/o el papel fotográfico. Sin embargo, debido a la complejidad que representa el procesamiento de datos este tipo de medios, actualmente, la mayor parte de las señales que se utilizan son detectadas, almacenadas y procesadas de manera digital. Esto hace posible que puedan ser analizadas con diversas herramientas matemáticas, lo que permite entre varias cosas: reducir el ruido, filtrar la diversas componentes, clasificar la señal, convertirla a otros espacios vectoriales y/o extraer sus propiedades

características. La Figura 5.1 muestra un ejemplo de señal analógica correspondiente al registro sísmico del sismo del 19 de septiembre de 1985, y sus réplicas obtenido por la estación CU, UNAM. En esta imagen se pueden ver una gran cantidad de sismos los cuales se sobreponen entre sí. Actualmente, las técnicas de procesamiento digital permiten extraer una amplia cantidad de información que hubiera sido impensable hace algunas décadas.

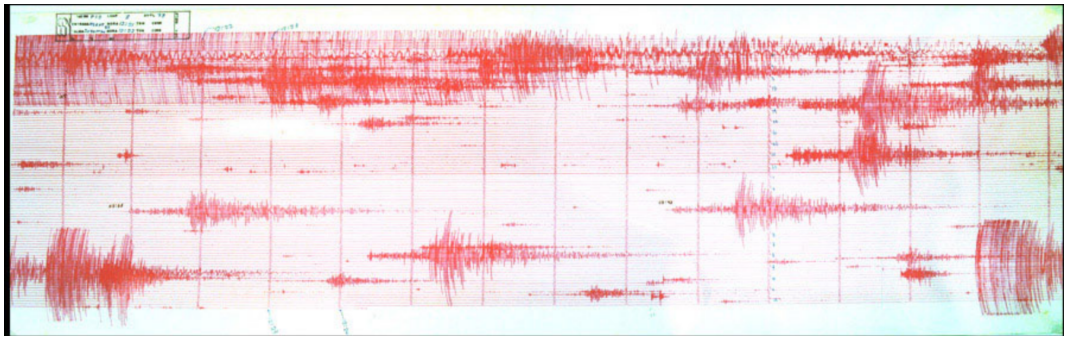

Figura 5.1: Registro sísmico analógico del 19 de septiembre de 1985. En este registro se observa el sismo principal junto con algunas de sus réplicas.

5.2. Conversión analógico digital

El primer paso para el procesamiento digital de señales es convertir una señal proveniente de un sensor analógico, en este caso un sismómetro, en una señal discretizada digital. Al dispositivo físico que realiza este proceso se le conoce como ADC (*analog-digital converter*). Este proceso transforma una función real de variable continua a una secuencia discreta en el tiempo, conocida como *bitstream*. Para llevar a cabo esta conversión, se deben seguir dos etapas: el muestreo y la cuantificación. Estos procesos consisten en transformar una señal de entrada a un voltaje mediante un transductor y posteriormente a una secuencia de números binarios mediante los cuales se almacena la señal. El proceso de muestreo y cuantificación de la señal se ilustra en la Figura 5.2.

5.2.1. Muestreo

Durante el muestreo se toma un instante de tiempo el voltaje de la señal de entrada proveniente del sensor, este proceso se llama *sample-and-Hold* (S/H) o *Track-and-Hold* (T/H). Cuando se congela la señal, el S/H abre y cierra una ventana para capturar el momento del voltaje en el borde de la señal de reloj, dando una señal de voltaje discreta como salida.

5.2.2. Cuantificación

La cuantificación toma como entrada la señal de voltajes del S/H. En este proceso se asigna un valor numérico a cada nivel del voltaje, mediante la búsqueda de un valor que corresponda a la amplitud de la señal en un rango de 2^n valores. El número asignado después se codifica como un número binario, la resolución[1] mide la precisión con la que se asignó un valor, se volverá más precisa mientras mayor sea el número de bits utilizado para representar la señal.

5.3. Propiedades de las señales en tiempo discreto

Al discretizarse una señal, esta tendrá algunas propiedades exclusivas de las señales discretas. Esto a su vez, establece una serie de limitaciones producto de haber reducido la cantidad de información de la señal original durante la etapa de cuantificación y muestreo. Idealmente, el n-ésimo valor de la secuencia discreta deberá de ser igual al n-ésimo valor de la secuencia continua. Es decir,

$$x[n] = x_a(nT),\ -\infty < n < \infty \tag{5.1}$$

Donde T, es el periodo de muestreo y su inversa es la frecuencia de muestreo.

[1] La resolución representa el número de bits usados por el ADC para codificar sus valores digitalizados.

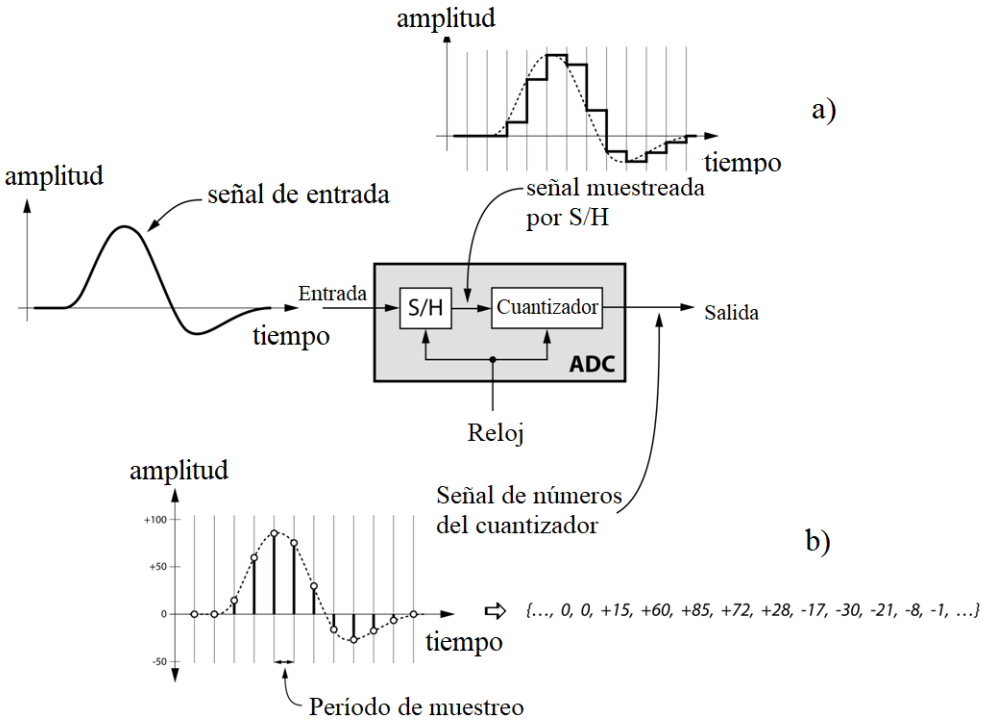

Figura 5.2: Proceso de la transformación de una señal análoga a digital. La sección a) ejemplifica el proceso de muestreo, la sección b) ejemplifica el proceso de cuantificación. Modificado de https://www.nutaq.com/blog/analog-digital-%E2%80%93-part-2-conversion-process. Último acceso: Julio, 2019.

Las señales en tiempo discreto, a diferencia de las señales continuas, contienen ciertos atributos específicos que determinan las técnicas de procesado o análisis de la señal que se van a emplear. Entre las propiedades de las señales discretas tenemos:

- **Duración de una señal**. Una señal en tiempo discreto es una secuencia de duración finita. Se considera que es igual a 0, para todos su valores fuera del intervalo $[X_1, X_2]$. En nuestro caso, se considera que la duración de cada sismo estará dado por una ventana de tiempo que inicia en el momento en que se registra la llegada de la primera onda proveniente del sismo (*onda P*),

y el momento en que la señal regresa al nivel de ruido registrado antes de la llegada del sismo.

- **Tasa de muestreo**. Es el número de muestras obtenidas por un intervalo de tiempo definido. Típicamente, las tasas de muestreo utilizadas en sismología van de 1-500 muestras por segundo.
- **Secuencia de lado derecho**. Secuencias de longitud finita donde sus valores son igual a 0, para todos los valores $n > n_0$ para algún entero n_0. Las señales analizadas en este proyecto se pueden clasificar como de lado derecho.
- **Secuencia de lado izquierdo**. Estas secuencias de longitud finita tienen las mismas propiedades que las secuencias de lado derecho, pero cumpliendo con $n < n_0$ para algún entero n_0.
- **Secuencias periódicas y aperiódicas**. Una señal periódica está definida por $x(n) = x(n + N)$, para todo n, es decir, la secuencia periódica no cambia en cada múltiplo de N muestras, al contrario de una señal aperiódica, cuyos valores jamás se repiten de forma secuencial. Los sismogramas son en general secuencias aperiódicas, sin embargo, como explicaremos más adelante para hacer uso de la transformada de Fourier, supondremos que las señales son periódicas pero con $T \to \infty$.

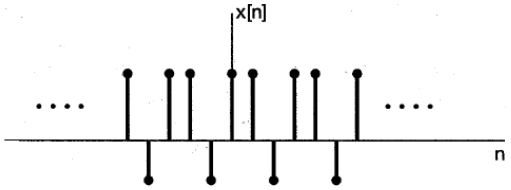

Figura 5.3: Ejemplo de señal periódica discreta con periodo $N_0 = 3$. Imagen: [Oppenheim et al., 1998]

- **Secuencia simétrica**. Son conocidas como señales pares o impares, y están definidas mediante la inversión de tiempo por la ecuación $x(n) = x(-n)$, para una señal par y para toda n, mientras que para una señal impar, $x(n) = -x(-n)$ y para toda n.

5.4. Análisis de Fourier

Para el presente trabajo ahondaremos en el tratamiento de la señal a través del análisis de Fourier. Este tratamiento nos permitirá determinar la existencia de sismos repetitivos, además de proporcionar un procesamiento adecuado de la señal para su clasificación. El análisis de Fourier fue propuesto por el matemático francés Jean Baptiste Joseph Fourier (1768 - 1830) quien en 1822 introdujo esta teoría como parte de sus experimentos de propagación del calor publicados en el libro "Teoría analítica del calor". Este planteamiento daría paso a lo que ahora conocemos como el análisis de Fourier. La idea principal consiste en la descomposición de una señal en una suma de funciones senos y cosenos. En una primera instancia, las serie de Fourier consiste en aproximar una señal periódica en un conjunto de funciones ortogonales[2]. Cuando se asume que el periodo de una función tiende al infinito (la señal que se está analizando es aperiódica), esta señal se puede mapear en otra señal continua compleja en el dominio de la frecuencia. A esto se le conoce como Transformada Directa de Fourier. El proceso inverso que se encarga de hacer corresponder una función $F(\omega)$ con valores complejos a la señal original el dominio del tiempo o de espacio, es por ende la Transformada Inversa de Fourier. Por otra parte, la Transformada de Fourier analiza el cambio de amplitud y fase mediante la representación fasorial de los números complejos de la forma,

$$Z = |Z| \exp i\theta. \tag{5.2}$$

Donde,

$$|Z| = \mathrm{Re(Z)}^2 + \mathrm{Im(Z)}^2, \tag{5.3}$$

y

$$\theta = \arctan \frac{\mathrm{Im(Z)}}{\mathrm{Re(Z)}}. \tag{5.4}$$

Podemos definir entonces, a la serie de Fourier como una sumatoria infinita de funciones más simples, senos y cosenos. Cualquier función periódica con período T,

[2]Dos funciones son ortogonales, si el producto punto es cero.

puede expresarse de la forma:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}[a_n \cos\frac{2n\pi}{T}t + b_n \sin\frac{2n\pi}{T}t] \tag{5.5}$$

siendo a_n y b_n los coeficientes de Fourier de $f(t)$, definidos como,

$$\begin{aligned} a_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)cos(nx)\,dx, \\ b_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)sen(nx)\,dx. \end{aligned} \tag{5.6}$$

El valor de los coeficientes a_n y b_n indican el *peso* que tiene cada uno de los pares de funciones seno y coseno para una determinada frecuencia angular, n. Por ejemplo, si encontramos que los valores de a_n y b_n son iguales o cercanos a cero para valores pequeños de n, esto significa, que nuestra señal contiene relativamente bajas frecuencias. Si por el contrario, estos valores fueran muy altos podemos inferir que nuestra señal es rica en bajas frecuencias. El concepto de **Transformada de Fourier** surge de obtener una distribución continua de valores de a_n y b_n, esto se logra asumiendo que la señal es periódica, pero con periodo $T \to \infty$.

5.5. Transformada de Fourier

Podemos decir que la Transformada de Fourier es un mapeo de una función en el dominio del tiempo a una función en el dominio de la frecuencia. La Transformada de Fourier calcula la estructura periódica de una señal en forma de conjunto de sinusoidales, calcula la potencia de las frecuencias para obtener el espectro de la señal[3], donde cada pico del espectro representará una onda sinusoidal que al ser sumada en todo el intervalo se obtendrá una señal aproximada de la señal original, ver Figura 5.4. Así, el espectro generado por la Transformada de Fourier a partir de una función periódica, es un conjunto de frecuencias, entonces la Transformada de

[3]Este espectro, resultado de la Transformada de Fourier, está caracterizado por las amplitudes en frecuencia de la señal

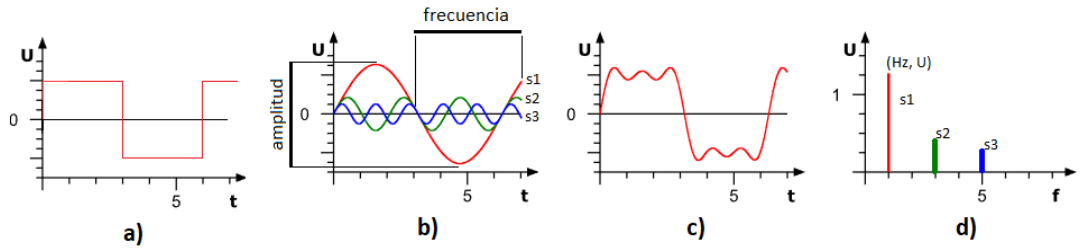

Figura 5.4: Ejemplo de la interpretación del proceso de una Transformada de Fourier. La figura (a) es la gráfica de la función cuadrada original en el dominio del tiempo. La figura (b) representa el espectro de ondas sinusoidales, descrito en las series de Fourier. La figura (c) representa la suma de las ondas sinusoidales de la señal original. La figura (d) es la densidad de frecuencia de las ondas, es decir, el resultado gráfico de la Transformada de Fourier, para el cual, cada pico se refiere a la frecuencia en el tiempo de cada onda sinusoidal.

Fourier de una función no periódica, produce un espectro de frecuencias continuo. La Transformada analiza una señal en los componentes de la frecuencia para señales periódicas en intervalos finitos, si se define una señal aperiódica en un intervalo finito, se pierde información acerca de la señal.

Definimos la Transformada de Fourier de la siguiente forma:

$$\mathcal{F}(f(t)) = \int_{-\infty}^{\infty} f(n)e^{i\omega t}dt. \tag{5.7}$$

Donde ω se conoce como la frecuencia angular, $f(t)$ es una función continua en el tiempo definida para todos los reales, e i corresponde al número imaginario, $\sqrt{-1}$. Note que la Ecuación 5.7 es análoga a la Ecuación 5.6 en donde se han remplazado las funciones senos y cosenos por una función exponencial imaginaria a través de la Ecuación de Euler[4]. La deducción formal de este resultado puede encontrarse fácilmente en la mayor parte de los libros de texto referente a la Transformada de Fourier [Rojas et al., 2014].

[4]La Ecuación de Euler establece que $e^{i\omega t} = cos(\omega t) + isin(\omega t)$.

5.6. Propiedades de la Transformada de Fourier

La Transformada de Fourier cumple con un conjunto de propiedades que permite manipular la señal ya sea en el dominio del tiempo o de la frecuencia. Esto permite modificar la señal de acuerdo a las necesidades de procesamiento sin necesidad de anti-transformar la señal al dominio original. Además, estas propiedades permiten simplificar muchas de las operaciones que de otra forma requerirían un mayor tiempo de cómputo. A continuación se describen las principales propiedades de la Transforma de Fourier.

1. **Simetría par e impar**. Cumple las mismas propiedades que la simetría de las series de Fourier. Definida por la función:

$$\begin{aligned} \mathcal{F}f(-\omega) &= \mathcal{F}^{-1}f(\omega) \\ \mathcal{F}^{-1}f(-t) &= \mathcal{F}f(t) \end{aligned} \tag{5.8}$$

   Es decir, existe una simetría en los valores del espectro de Fourier con la función original. Esto quiere decir que si se reemplaza ω por $-\omega$ en la ecuación de Fourier, se está tomando la inversa de la Transformada. Entonces, al reemplazar t por $-t$ en la ecuación de la inversa de Fourier, se obtiene la Transformada de Fourier.

2. **Linealidad**. Esta propiedad está definida por:

$$\mathcal{F}(\alpha f(t) + \beta g(t)) = \alpha F(\omega) + \beta G(\omega). \tag{5.9}$$

   Esto significa, que la suma ponderada de dos funciones $f(t)$ y $g(t)$ en el tiempo, será igual a la suma ponderada de las transformadas de dichas funciones.

3. **Desplazamiento en el tiempo**. Se refiere al cambio ocurrido en la fase[5] de la Transformada de Fourier, al ser efectuado en la variable t (que se refiere al

[5]Se define a la fase de una señal, como la diferencia de tiempo de dos señales en el mismo instante de ambas señales.

tiempo, es decir, un retraso en la señal), mientras que la magnitud permanece igual. Definida por la función:

$$f(t \pm b) \overset{\mathcal{F}}{\longleftrightarrow} e^{\pm 2\pi i \omega b} F(\omega). \tag{5.10}$$

Esta propiedad es importante para nuestro proyecto ya que nos permitirá *empalmar* nuestras funciones de tal forma que el valor de los coeficientes de correlación y coherencia espectral sean máximos.

4. **Escalamiento**. Definido por la función $f(at) \overset{\mathcal{F}}{\longleftrightarrow} \frac{1}{|a|} F(\frac{\omega}{a})$ describe el efecto que produce *escalar* la variable tiempo por una constante a en dominio de tiempo. Si la función en el dominio de tiempo sufre una *compresión* o *estiramiento* en el dominio del tiempo su correspondiente Transformada sufrirá una disminución/ampliación en su amplitud así como un escalamiento inversamente proporcional en la frecuencia ω.

5. **Identidad de Parseval.** La identidad define una relación entre la energía de la señal en el tiempo y la energía de la señal en la frecuencia.

$$\int_{-\infty}^{\infty} |f(t)|^2 dt = \int_{-\infty}^{\infty} |\mathcal{F}(\omega)|^2 d\omega \tag{5.11}$$

En ingeniería, la integral sobre todo el intervalo del cuadrado de una función se le conoce como **potencia**. Por ejemplo, si $f(t)$ representa el voltaje en una resistencia, la integral será proporcional a la resistencia disipada por dicha resistencia. El Teorema de Parseval establece que dicho cálculo de la potencia puede realizarse de manera indistinta ya sea en el dominio del tiempo o la frecuencia. Para nuestro estudio, el cálculo de la energía de la señal (*sismograma*) es necesario para la normalización de los coeficientes de correlación al comparar dos señales entre sí.

5.7. Transformada Discreta de Fourier

Para poder calcular las componentes espectrales de una señal proveniente de un digitalizador[6], es necesario obtener una versión discreta de la Ecuación 5.7, y que cumpla con las propiedades mencionadas en la sección anterior. En este caso asumiremos que las muestras de la señal de entrada tienen un espaciamiento uniforme entre sí y que su duración es finita. Este es el caso más común que encontramos en el procesamiento de señales físicas. La ecuación que permite discretizar la Transformada de Fourier es la siguiente:

$$\begin{aligned} X_k &= \sum_{n=0}^{N} x_n \, e^{-\frac{i2\pi}{N} kn} \\ &= \sum_{n=0}^{N} x_n \left[\cos\left(\frac{2\pi n}{N}\right) + i \sin\left(\frac{2\pi n}{N}\right) \right]. \end{aligned} \tag{5.12}$$

Donde, $x_n = \{x_0, x_1, \ldots, x_{N-1}\}$ es la señal de entrada en tiempo, y X_k corresponde a los valores de la Transformada Discreta de Fourier. Como veremos más adelante la representación en frecuencia permite realizar una gran cantidad de operaciones sobre la señal de entrada de forma que es posible extraer las características de nuestro interés. Sin embargo, la obtención de la Transformada Discreta de Fourier, a través de aplicar directamente la Ecuación 5.12 es computacionalmente ineficiente, no obstante, al aplicar las diversas propiedades de la DFT se puede eficientar notablemente su cálculo. Con esto es posible obtener la FFT de un conjunto amplio de señales de forma simultánea con lo que se logra una aceleración de 10 veces con respecto al CPU.

[6]Se conoce como digitalizador al dispositivo físico que realiza los procesos de conversión analógico digital, muestreo, almacenamiento y en algunos casos transmisión de la datos proveniente de un sensor analógico.

5.8. Transformada rápida de Fourier(FFT)

Uno de los métodos para calcular la Transformada discreta de Fourier (DFT) es el algoritmo de la Transformada Rápida de Fourier (FFT, *Fast Fourier Transform*). Este algoritmo hace uso de las propiedades de simetría en el cálculo de la Transformada discreta de Fourier (Ver Ecuación 5.13) de forma que reduce significativamente el número de operaciones matemáticas necesarias para llevar a cabo la transformación. Este algoritmo optimiza la implementación directa de la Transformada Discreta de Fourier, es decir, FFT obtiene el mismo resultado que la implementación directa de la Transformada discreta. La Transformada de Fourier discreta es una representación de Fourier para señales de duración finita, está definida por la siguiente ecuación:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^k \text{ para } k = 0, 1, \ldots, N. \tag{5.13}$$

Donde $W_N^k = e^{-j\frac{2\pi}{N}}$ representa el factor de Twiddle[7], y donde j es la unidad imaginaria. Dada la complejidad de la ecuación para valores muy grandes, tenemos que, por ejemplo, para valores de $n = 2^{30}$ las operaciones serían 2^{60}, si cada operación tomara $1ns$, el cálculo tardaría más de 13000 días aproximadamente [Schmidt, 2013]. Para valores mayores de algún N, el cálculo es muy lento, dado este problema surge entones la implementación de la Transformada de Fourier, que elimina muchos cálculos iterativos que se implementan en el cálculo directo de la Transformada Discreta.

La importancia del análisis discreto radica en que es prácticamente y teóricamente imposible representar valores continuos en una computadora. Esto debido a que todas las computadoras tienen un espacio de almacenamiento limitado, es decir, la memoria puede guardar números finitos de bits, que son un conjunto finito de

[7]Son coeficientes usados durante operaciones en el algoritmo radix para combinar recursivamente Transformadas de Fourier discretas más pequeñas. El conjunto de W valores se repite cada vez acorde al número de muestras. El factor de Twiddle describe el vector de rotación.

cadenas de bits, por lo que solo se puede trabajar con números reales.

El algoritmo de FFT más común se basa en Radix-2 FFT, otro algoritmo que opera los N puntos de la DFT y los divide a la mitad obteniendo dos grupos de $N/2$ cada una y así sucesivamente. Este algoritmo es parte del cálculo de FFT. En este trabajo hicimos uso de la librería de cuFFT[8] de CUDA, la cual permite el cálculo eficiente de la FFT considerando las ventajas que ofrece la arquitectura del GPU.

5.8.1. Radix-2 FFT

La Transformada rápida de Fourier se basa en este algoritmo para calcular la Transformada discreta de Fourier, opera descomponiendo una señal de N puntos de dominio tiempo en otra señal de N señales de dominio de tiempo de un solo punto. Los N puntos individuales resultantes deben reconstruirse en una frecuencia de N puntos, esto se logra usando cálculos de mariposa, ver Figura 5.5. En cada etapa, el algoritmo realiza este método con dos puntos, para posteriormente aplicar una reducción de orden con la función W_N^R, donde N = número de puntos y R = etapa.

Tenemos entonces que al dividir la TFD en dos mas pequeñas:

$$\begin{aligned} X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} &= \sum_{n=0}^{N/2-1} x[2n] W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1] W_N^{(2n+1)k} \\ X[k] &= \sum_{n=0}^{N/2-1} x[2n] (W_N^2)^{nk} + W_N^k \sum_{n=0}^{N-1} x[2n+1] (W_N^2)^{nk} \end{aligned} \tag{5.14}$$

[8] https://developer.nvidia.com/cufft

Sabemos por el factor de Twiddle que $W_N^2 = e^{-j\frac{2\pi}{N}2} = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$, expresamos entonces la ecuación en una suma de $N/2$ muestras.

$$X[k] = \sum_{n=0}^{N/2-1} x[2n](W_{N/2})^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1](W_{N/2})^{kn} \tag{5.15}$$

En este resultado siguen siendo dos Transformadas discretas, entonces podemos volver a aplicar el método de división para obtener N/4 muestras de Transformadas discretas, hasta obtener Transformadas de 1 muestra.

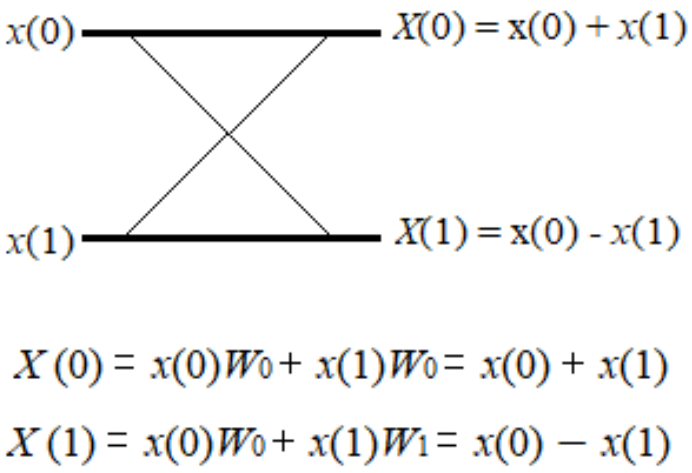

Figura 5.5: Cálculo de FFT en dos puntos representado por el diagrama "mariposa". Se muestra la última etapa del algoritmo FFT, donde $x(n)$ es la secuencia y $W^n = cos\frac{2}{N} - jsen\frac{2\pi}{N}$, que se le denomina: constante de Twiddle. Este factor representa la simetría y periodicidad del factor de fase.

Para este método tomamos un ejemplo de un arreglo de 8 puntos, expresado con la secuencia $x = \{1, 1, 1, 1, 0, 0, 0, 0\}$ para la cual calcularemos su DFT ejemplificando el método Radix-2, apoyándonos de la Figura 5.6. Tenemos entonces que para cada valor de $x(n) = W^n$,

$$W^0 = 1 \tag{5.16}$$
$$W^1 = e^{-\frac{j2\pi}{8}} = \cos\frac{\pi}{4} - j\sin\frac{\pi}{4} = 0.707 - j0.707 \tag{5.17}$$
$$W^2 = e^{-\frac{j2\pi}{4}} = \cos\frac{\pi}{2} - j\sin\frac{\pi}{2} = -j \tag{5.18}$$
$$W^3 = e^{-\frac{j2\pi 3}{8}} = \cos\frac{3\pi}{4} - j\sin\frac{3\pi}{4} = -0.707 - j0.707 \tag{5.19}$$
$$\tag{5.20}$$

Las salidas X(n), para la etapa 1 serían entonces:

$$x(0) + x(4) = 1 \qquad \rightarrow X(0) \tag{5.21}$$
$$x(1) + x(5) = 1 \qquad \rightarrow X(1) \tag{5.22}$$
$$x(2) + x(6) = 1 \qquad \rightarrow X(2) \tag{5.23}$$
$$x(3) + x(7) = 1 \qquad \rightarrow X(3) \tag{5.24}$$
$$[x(0) + x(4)]W^0 = 1 \qquad \rightarrow X(4) \tag{5.25}$$
$$[x(1) + x(5)]W^1 = 0.707 - j0.707 \qquad \rightarrow X(5) \tag{5.26}$$
$$[x(2) + x(6)]W^2 = -j \qquad \rightarrow X(6) \tag{5.27}$$
$$[x(3) + x(7)]W^3 = -0.707 - j0.707 \qquad \rightarrow X(7) \tag{5.28}$$

Siguiente en la etapa 2, tenemos las salidas:

$$X(0) + X(2) = 2 \qquad \rightarrow X(0) \tag{5.29}$$
$$X(1) + X(3) = 2 \qquad \rightarrow X(1) \tag{5.30}$$
$$[X(0) + X(2)]W^0 = 0 \qquad \rightarrow X(2) \tag{5.31}$$
$$[X(1) + X(3)]W^2 = 0 \qquad \rightarrow X(3) \tag{5.32}$$
$$X(4) + X(6) = i - j \qquad \rightarrow X(4) \tag{5.33}$$
$$X(5) + X(7) = -1.414j \qquad \rightarrow X(5) \tag{5.34}$$
$$[X(4) + X(6)]W^0 = 1 + j \qquad \rightarrow X(6) \tag{5.35}$$
$$[X(5) + X(7)]W^2 = -1.414j \qquad \rightarrow X(7) \tag{5.36}$$

Usamos nuevamente, las salidas de la etapa anterior como entrada para la siguiente y última etapa:

$$X(0) = X(0) + X(1) = 4 \tag{5.37}$$
$$X(4) = [X(0) + X(1)]W^0 = 0 \tag{5.38}$$
$$X(2) = X(2) + X(3) = 0 \tag{5.39}$$
$$X(6) = [X(2) + X(3)]W^0 = 0 \tag{5.40}$$
$$X(1) = X(4) + X(5) = 1 - 2.414j \tag{5.41}$$
$$X(5) = [X(4) + X(5)]W^0 = 1 + 0.404j \tag{5.42}$$
$$X(3) = X(6) + X(7) = 1 + 0.404j \tag{5.43}$$
$$X(7) = [X(6) + X(7)]W^0 = 1 + 2.41j \tag{5.44}$$

Donde los resultados de esta etapa representan los resultados finales para el proceso de Radix-2.

5.9. Densidad espectral de potencia

La densidad espectral de potencia $P_{xx}(\omega)$ (PSD) de una serie de tiempo x(t), describe la distribución de potencia de cada una de sus componentes es frecuencia. Como vimos anteriormente, una señal se puede descomponer mediante la Transformada de Fourier, en una señal en el dominio de la frecuencia. Cuando la energía de una señal se concentra en un intervalo finito de frecuencias es posible calcular su densidad espectral de potencia, la cual es un indicador de la energía transportada por la señal por unidad de tiempo. Entonces, la densidad espectral de potencia se obtiene de la sumatoria (integración) del cuadrado de todas la componentes espectrales de la siguiente forma,

$$P_{xx}(f) = \int_{-\infty}^{\infty} |X(\omega)|^2 \, d\omega. \tag{5.45}$$

Donde $X(\omega)$ es la Transformada de Fourier, ya sea continua o discreta de la señal $x(t)$. Por otra parte, es posible calcular el espectro cruzado de potencia $P_{xy}(\omega)$, entre

dos señales $x(t)$ y $y(t)$. De la forma,

$$P_{xy} = \int_{-\infty}^{\infty} |X(\omega)||Y(\omega)|\, \omega \tag{5.46}$$

La cual se puede estimar en el dominio del tiempo como,

$$P_{xy} = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} x(\tau) y(tau - t) d\tau \right] e^{-i\omega t}\, dt. \tag{5.47}$$

Esta última expresión se puede obtener fácilmente de aplicar el Teorema de Convolución[9]. En nuestro caso, el cálculo de la potencia de la señal y del espectro cruzado de potencia servirán para el normalizado tanto del coeficiente de correlación como de la coherencia espectral, que se definirán a continuación.

5.10. Coeficiente de Correlación

El coeficiente de correlación establece el grado de correlación entre dos variables aleatorias. Si dos variables están perfectamente correlacionadas, se comportan de manera sincronizada y simétrica. Es decir, un incremento en una de las variables implica que la segunda variable aumentará de igual forma, como consecuencia el coeficiente de correlación será igual a 1. Si por el contrario, las variables se comportan de manera sincronizada pero antisimétrica, el cambio en una de las variables da como resultado un cambio de igual magnitud pero en sentido opuesto en la otra, entonces el coeficiente de correlación será igual a -1. En el caso, de que el cambio en una variable no puede ser determinado a partir de la otra, entonces el coeficiente de correlación es igual a 0. Matemáticamente, el coeficiente de correlación se puede escribir como,

$$CC = \frac{R_{xy}}{N\sqrt{P_{xx}P_{yy}}} \tag{5.48}$$

Donde R_{xy}, corresponde a la covarianza de las señales $x(t)$ y $y(t)$; mientras que P_{xx} y P_{yy} corresponden a sus respectivas densidades espectrales de potencia, las

[9]El Teorema de Convolución establece que la multiplicación de dos señales en el dominio de la frecuencia equivale a la convolución de las señales en el dominio del tiempo. Dicho teorema puede ser encontrado fácilmente en la literatura del tema

cuales permiten la normalización del coeficiente de correlación. Debido a que las señales analizadas en este trabajo, pueden tener ligeras desviaciones al momento de seleccionar el inicio de la señal, es necesario calcular el punto donde ocurre el máximo del coeficiente de correlación. Con esto, es posible determinar cual es la corrimiento en tiempo necesario que permite *empalmar* de manera óptima las dos señales.

5.11. Coherencia espectral

La coherencia es el grado de relación entre dos series de tiempo x_m y y_m, en función de la frecuencia. Es decir, es una función que estima la consistencia de amplitud y fase para un par de secuencias en cada banda de frecuencia. Definida por la ecuación:

$$C_{xy} = \frac{P_{yx}(s)}{\sqrt{P_{yy}(s)P_{xx}(s)}} \tag{5.49}$$

Donde P_{xy} es la densidad espectral de potencias de las señales $x(n)$ e $y(n)$ (también llamada densidad espectral de potencia cruzada), y P_{yy} y P_{xx} es la PSD de $y(n)$ consigo misma y la PSD de $x(n)$ consigo misma, respectivamente (también llamado autoespectro). La coherencia se basa en la consistencia de fase, esto es si dos señales tiene diferente fase, la alta coherencia se da cuando la diferencia de fase de las señales es constante para cada frecuencia, la coherencia mide cuando las señales se relacionan mediante una transformación lineal, esto es, se tiene una razón de amplitud constante y de desplazamiento de fase. Para este estudio limitados la banda de frecuencia mediante el uso de filtro paso bandas (como se explicará en la siguiente sección) de 1-8Hz, esto debido a que es la banda de frecuencia donde se concentra la mayor parte de la energía para los sismos que estamos analizando.

5.12. Filtrado de señales

Una de las operaciones en las que se utiliza comúnmente la FFT es el filtrado. Las señales obtenidas a partir del registro de un fenómeno físico, estarán afectadas por una diversidad de procesos que pueden ser introducidos ya sea por la naturaleza misma del sistema que se está registrando, o por las características propias del instrumento (ruido electrónico, ambiental, etc.). Con el objetivo de aislar la señal de interés y mejorar la calidad de la misma es posible aplicar un filtro, el cual consiste en un sistema que toma una señal de entrada y la transforma en una señal de salida a la cual se le han removido ciertas características no deseadas de la señal original. A continuación se describen los tipos más comunes de filtros:

- **Filtro paso bajo**. Este tipo de filtro elimina del espectro en frecuencia aquellos componentes espectrales de la señal que sobrepasen en amplitud una frecuencia dada $|w| \geq w_0$. En sismología, este filtro se utiliza para separar la señal producida por las ondas superficiales (bajas frecuencias) de las señales producidas por ondas de cuerpo o señales urbanas de alta frecuencia.

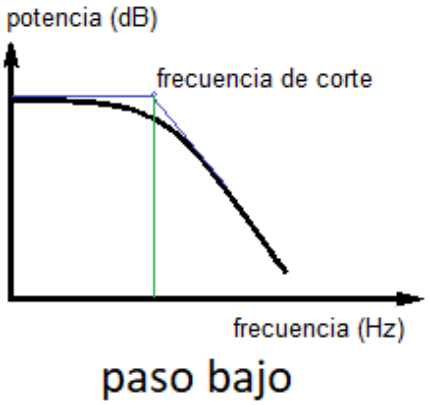

Figura 5.7: A medida que la frecuencia aumenta, la potencia que deja pasar tiende a 0.

El filtro está definido por:

$$H_{w_0}(w) = \begin{cases} e^{-iwt_0}, & |w| < w_0, \\ 0, & \text{otro caso} \end{cases} \tag{5.50}$$

Tenemos que: $H(w) = \hat{h}(w)$. Donde $H(w)$ es la función de transferencia del sistema y $\hat{h}(w)$ es la salida en el dominio de la frecuencia. Tenemos entonces que:

$$h_{w_0}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} H_{w_0}(w)e^{iwt}dt = \frac{1}{2\pi}\int_{-w_0}^{w_0} e^{iw_0t-t_0}dw = \frac{sin(w_0(t-t_0))}{\pi(t-t_0)} \tag{5.51}$$

Definiendo nuestro filtro como: $L_{w0}(x)(t) = \int_{-\infty}^{\infty} x(s)\frac{senw_0n}{\pi n}ds$

- **Filtro paso medio**. Este tipo de filtro deja pasar solo las frecuencias dentro de un rango determinado por lo que tiene múltiples aplicaciones. Por ejemplo, en la transmisión de señales de radio, este filtro permite seleccionar una estación de radio determinada por una frecuencia de portadora. Por ejemplo, un filtro de radiofrecuencia alrededor de la frecuencia de 102.5MHz, en la ciudad de Morelia, hará que el escucha sintonice *Radio Ranchito*.

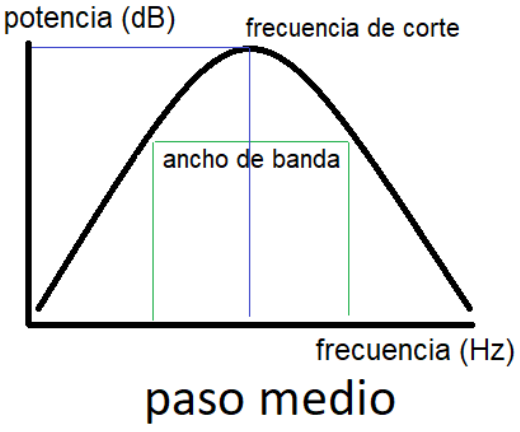

Figura 5.8: A medida que la señal se acerca a la frecuencia deseada, se conservan las componentes espectrales de la señal cercana a ese punto.

El filtro esta definido en:

$$H_{w_c,w_0}(w) = \begin{cases} e^{-iwt_0}, & \text{si } \max(|w - w_c|, |w + w_c|) \leq w_0 \\ 0, & \text{otro caso} \end{cases} \tag{5.52}$$

- **Filtro paso alto**. Este filtro reduce aquellos componentes espectrales de baja frecuencia a una frecuencias de corte. En sismología, este filtro se utiliza para mejorar la relación señal a ruido de sismos pequeños con contenido alto de frecuencias, cuyas amplitudes apenas superan el nivel de ruido.

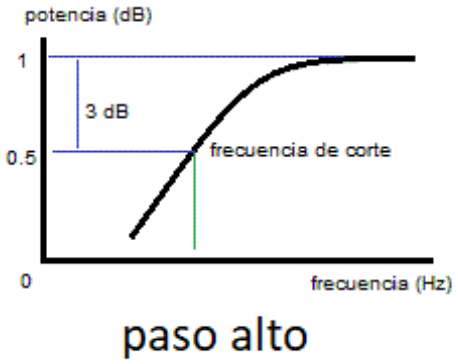

Figura 5.9: A medida que la frecuencia aumenta, el filtro conserva aquellas componentes espectrales por encima de la frecuencia de corte.

Para este proyecto utilizaremos un filtro paso banda de $1 - 8Hz$ que es el rango de frecuencias donde se concentra la mayor parte de la energía de los sismos de interés. Por otra parte, también reduce el efecto producido por las ondas de superficie proveniente de eventos lejanos de mayor magnitud. Frecuencias mayores a los $8Hz$ están asociadas principalmente al ruido urbano proveniente del paso de vehículos y personas.

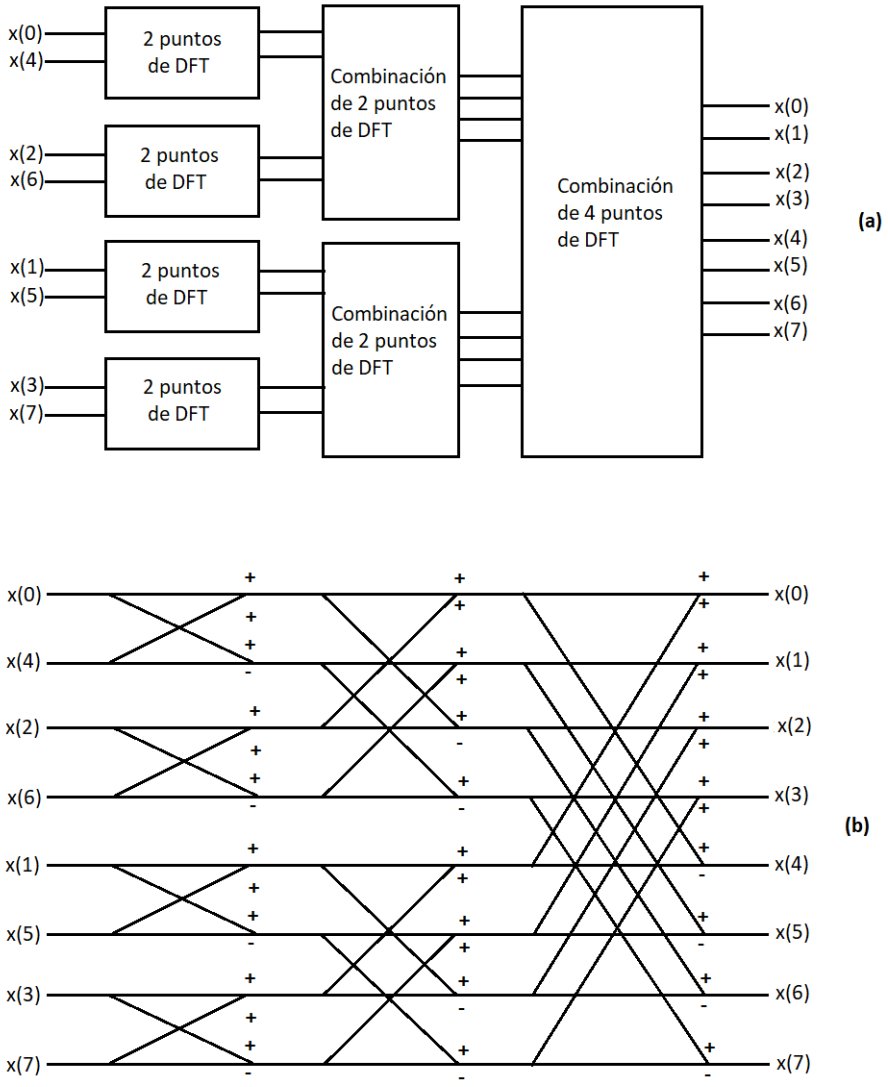

Figura 5.6: FFT de 8 puntos usando Radix-2. a) Esquema general. b) Operaciones aritméticas. Imagen modificada de Matias Zañartu (2012), Procesamiento digital de Señales con Aplicaciones

CAPÍTULO 6

METODOLOGÍA

Contenido

A lo largo de este capítulo, describiremos el método desarrollado para el procesamiento y análisis de los datos. Desde la obtención de los datos y su pre-procesamiento en CPU hasta la implementación del algoritmo en GPU. Los resultados obtenidos en cada proceso nos dan información sobre el estado de los datos y los métodos que se deberían implementar, a partir de la teoría explicada a lo largo de este proyecto. Tras saber que método utilizar, este se debería aplicar para conocer el próximo estado de los datos. También se describen los recursos utilizados sobre los cuales se ejecutó el algoritmo, así como su desempeño.

La obtención de las secuencias de datos se realizó mediante los registros de las estaciones del SSN (Servicio Sismológico Nacional) entre el año 2001 y el año 2018, a través del protocolo STP (*Seismogram Transfer Protocol*[1]), mediante el cual se recopiló un conjunto de diez mil secuencias de señales sísmicas. Para su procesamiento a través de la Transformada de Fourier, se utilizó la librería *cudaFFT*. Para nuestro propósito, realizamos la siguientes suposiciones: (1) la señal original es continua en el tiempo, por lo anterior, es necesario discretizarla, para un número finito de valo-

[1] https://scedc.caltech.edu/research-tools/stp/STPdocumentation.html

res equiespaciados, este proceso se realiza directamente en la estación sismológica, los datos utilizados ya se encuentran muestreados a una tasa de 100 muestras por segundo. (2) El espectro de la señal está afectada por una cierta cantidad de ruido. Entendiéndose como ruido, todas aquellas componentes del espectro que no sean producidas por la señal de interés, en este caso el sismo que se quiere analizar.

Los modelos de GPU usados en el proyecto se obtuvieron pensando en la ejecución de la arquitectura CUDA, se analizaron dos tipos de tarjetas GPU de la marca NVIDIA: (1) Una tarjeta gráfica externa Geforce GTX 1070, y una interna para Laptop modelos NVIDIA 1080. En la Figura 6.1 se muestra el diagrama de flujo, identificando tres partes: obtención de datos y resultados, preprocesamiento de información (Entrada a GPU y salida de CPU), procesamiento en GPU. Donde estas partes del algoritmo no son necesariamente lineales, sino que oscilan entre estos procesos y son dependientes entre si para la obtención de información. Además podemos observar que los procesos más costosos computacionalmente son ejecutados dentro de la sección de la GPU. Se usó la versión 9 de CUDA (V. 9.2) basándose en código C (pudiendo soportar C++ y Fortran), para el procesamiento en paralelo.

En las siguientes secciones se describe la implementación de las funciones utilizadas en el procesamiento de los datos y las pruebas de desempeño del algoritmo tanto en tiempo de ejecución como en precisión.

6.1. Preprocesamiento de datos

Las secuencias obtenidas se filtraron usando el software SAC (*Seismic Analysis Code*), escrito en C. Este software es de uso generalizado en sismología y permite realizar operaciones aritméticas básica así como obtener espectros de frecuencia entre otras cosas. Las señales se almacenan en archivos separados con formato ".sac", esto son archivos binarios definidos por secciones, donde el encabezado tiene 158 palabras

Cabecera	Primera sección	Segunda sección (opcional)
Inicio: 0	Inicio: 158 words	Inicio: 158 words + NTPS
longitud: 158 words	longitud: NPTS	longitud: NPTS
	• Variable dependiente • Amplitud • Parte real	• Variable independiente • Fase • Parte Imaginaria

Tabla 6.1: Estructura de los archivos binarios, SAC. NPTS representa el número de puntos. Tabla modificada del manual de usuario de SAC. V 101.6, Noviembre 17, 2014.

de 32 bits de longitud y el resto pertenece a la sección de datos. El encabezado maneja campos de punto flotante, enteros y de caracteres. Los datos tienen una sola sección que contiene la variable dependiente. En los diferentes tipos de datos, como los datos espaciados de manera desigual, la primera sección de datos contiene la variable dependiente y la segunda contiene la variable independiente.

Para poder usar la tarjeta es necesario descargar e instalar el kit de desarrollo CUDA de la página oficial de NVIDIA (www.developer.nvidia.com). Se podría pensar que la versión más reciente (V10.1, Feb. 2019) es la mejor opción, sin embargo, es necesario decir que la versión de CUDA depende de nuestra tarjeta NVIDIA dada la capacidad de cálculo (NVIDIA lista las tarjetas y su compatibilidad adecuada con la versión de CUDA, www.developer.nvidia.com/cuda-gpus). Como se explicó en una sección anterior sobre el uso de las GPUs, cada arquitectura tiene características que versiones anteriores no tienen y que no se aprovecharán por la versión adecuada. La versión usada para este proyecto en ambas tarjetas gráficas fue CUDA 9 (V9.1, Dic. 2017) y el pre-procesamiento de las señales sísmicas fueron filtradas usando SAC, marcando la detección de ondas P (**P**rimarias, onda en dirección de la propagación). La incorporación de la primera parte del proceso en el algoritmo, implementado en CUDA, es lo que llamaremos "*Host*"que como se mencionó en capítulos anteriores, hace referencia al CPU y su memoria, capaz de administrar la memoria tanto del *Host* como del *Device* (refiriéndose al GPU) y gestiona la invocación al *kernel* (función en CUDA), la definición de los hilos y cómo serán agrupados. En la Figura 3.2,

del capítulo 3, puede diferenciarse la implementación de una función en C contra un *kernel*.

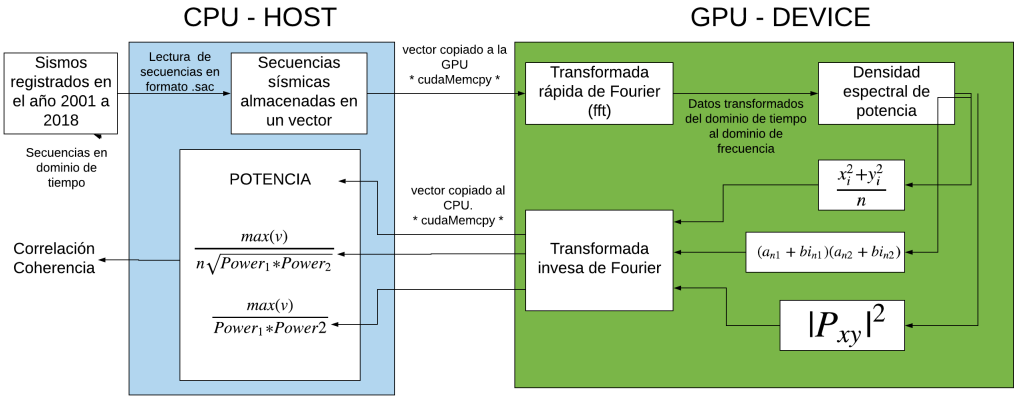

Figura 6.1: Diagrama de los procesos implementados en el algoritmo.

La lectura de los archivos fue individual, cada archivo contaba con mil veinticuatro datos de punto flotante, para agruparse en un único vector que representaba N entradas (señales sísmicas). Por ejemplo, para un $N = 5$ con $NPTS = 1024$, donde $NPTS$ es el número de puntos o datos por entrada, podemos ver la Tabla 6.2. La organización de los datos se realizó copiando el vector de datos del *Host* al *Device*. Para esto se utilizó una implementación CUDA del memcpy de C, que copia los valores de los bytes del bloque de memoria al que apunta la fuente, en el bloque de memoria al que apunta el destino.

cudaMemcpy(void* dst, void* src, size_t, kind)

- dst - Destino de la dirección de memoria
- src - Fuente de la dirección de memoria
- size_t - Tamaño de los bytes a copiar
- kind - Tipo de transferencia

- *Host*To*Device*
- *Host*To*Host*
- *Device*To*Device*
- *Device*To*Host*

I1	0,1, ... ,1023	0,1, ... ,1023	0,1, ... ,1023	0,1, ... ,1023	0,1, ... ,1023
I2	0,1, ... ,1023	1024, ... ,2047	2048, ... ,3071	3072, ... ,4095	4096, ... ,5119
	secuencia 1	secuencia 2	secuencia 3	secuencia 4	secuencia 5

Tabla 6.2: Se muestra un ejemplo de acomodo para cinco secuencias de entrada en un único vector. I1, representa el índice por secuencia dentro del vector, mientras que I2 representa el índice general de las secuencias dentro del vector. También funciona para ver la identificación de hilos en CUDA. I1 representa el identificador de los hilos por bloque, I2 el identificador de los hilos globales y cada secuencia la podemos ver como un identificador de bloque.

6.2. Implementación de la Transformada Rápida de Fourier

Recordando el análisis de Fourier mencionado en capítulos anteriores, la Transformada de Fourier es una herramienta que lleva del dominio de tiempo de una secuencia al dominio de frecuencia, obteniendo como resultado un espectro de frecuencias que nos indica la distribución de magnitudes de cada frecuencia, así, podemos representar una función como una suma de sinusoidales. Ver Figura 5.4.

Dicho lo anterior, usamos una función ya implementada en CUDA de la Transformada rápida de Fourier, que permite crear un plan para Transformadas de hasta tres dimensiones, con cuatro argumentos.

cufftPlan1D() / cufftPlan2D() / cufftPlan3D()
cufftPlanxD(plan, nx, type, batch)

- plan. Plan apuntador a un objeto de tipo de dato cufftHandle, este tipo de dato se encarga de guardar y usar el plan que define.
- nx. Tamaño de la Transformada
- type. Tipo de transformación de datos
  - CUFFT_R2C de reales a complejos
  - CUFFT_C2R de complejos a reales
  - CUFFT_C2C de complejos a complejos
- batch. Número de secuencias

El tercer argumento (*type*) se refiere al tipo de dato al que pertenecen nuestras entradas y el tipo de dato de salida. Obtenemos como salida un vector de la misma longitud que definimos al inicio, N entradas por mil veinticuatro datos. Este vector de datos complejos, resultante de la Transformada, representa nuestra señal en el dominio de la frecuencia.

6.3. Implementación en GPU de la potencia espectral

La potencia es usada para caracterizar la intensidad de una señal, es decir nos dice la energía de una señal por unidades de tiempo. En nuestro caso, utilizamos la potencia con el fin de normalizar el coeficiente de correlación a un valor entre 0 y 1, donde 1 indica que la señal es idéntica a sí misma. Definimos las variables para la invocación del *kernel* (que se refiere a la función que va a ser ejecutada en la GPU). Recodermos que la estructura del modelo CUDA se basa en la agrupación de hilos en bloques y mallas. Para poder trabajar con esta información hacemos uso

del tipo `dim3`, el cual es un tipo vector entero que se usa en CUDA para definir las dimensiones de la malla de hilos para la invocación del *kernel*.

1. dim3 GridDim(x,y,z)
2. dim3 BlockDim(x,y,z)

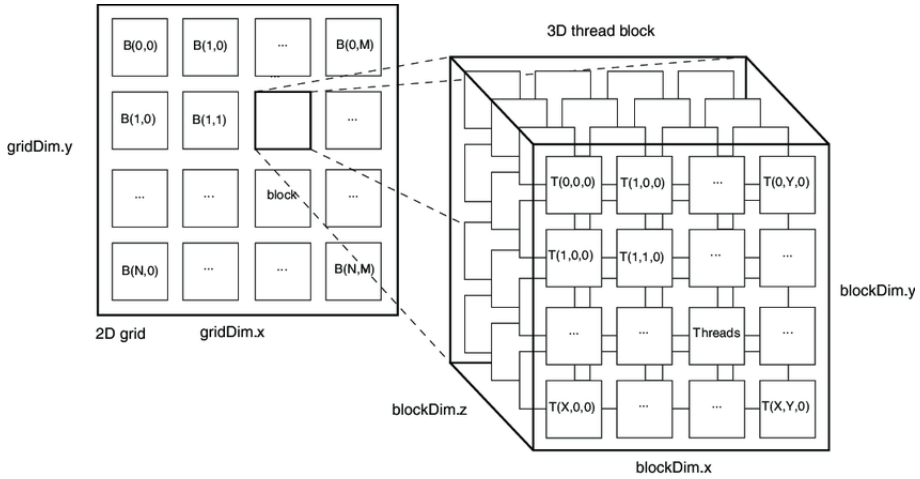

Figura 6.2: En la imagen podemos ver una malla definida en dos dimensiones de X*Y bloques y cada bloque dentro de la malla está definido en tres dimensiones, X*Y*Z hilos. Imagen extraída de "A survey on graphic processing unit computing for large-scale data mining", Alberto Cano (2018).

Las variables x, y y z indican la dimensión de la malla las cuales pueden ser de una, dos o tres dimensiones, pudiendo dar como entrada un solo argumento. Por ejemplo, `dim3 GridDim(4)` es igual a `dim3 GridDim(4,1,1)` que se puede ver como un vector en $\mathbb{R}^1$. Por su partedim3 `GridDim(4,4,1)` se puede representar como una matriz o un vector en $\mathbb{R}^2$ y `dim3 GridDim(4,4,4)` como un cubo o un vector en $\mathbb{R}^3$. Las dimensiones de la malla se basan en unidades que representan la cantidad de bloques y las dimensiones del bloque representan la cantidad de hilos por bloque. Ver Figura 6.2.

Con las definiciones hechas invocamos el *kernel* como cualquier función de C, pero con ciertas características intrínsecas. Especificamos el tamaño de la malla que vamos a ejecutar en cada *kernel*, en este caso usamos el tipo de dato dim3 para definir una malla de 3 dimensiones. Aunque estamos definiendo al *kernel* en el *Host*, este se llamará y ejecutará en el *Device*, por lo que su salida pertenecerá al *Device* y dado que los objetos del *Device* y el *Host* no se pueden comunicar, definimos una variable para nuestra salida reservando espacio de memoria en el *Device* con la función `cudaMalloc()`, que tiene el mismo propósito que la función de C/C++, `malloc()`.

Definimos entonces la función *Power*, que estará definida en el *kernel* con la palabra reservada `__global__` que indicará su ejecución en el *Device*, seguida por `void` (Los *kernel* no regresan una respuesta) y el nombre de la función. Cada hilo ejecutará un proceso del *kernel*, por lo que debemos elegir el orden de ejecución de los hilos. Cada hilo/bloque tiene un identificador para sus dimensiones, como podemos observar en la Figura 6.2.

Teniendo definida la ejecución de los hilos, usamos sus índices en términos globales para operar sobre el vector y calcular la potencia de todas las secuencias de señal en el vector. Cada elemento del vector es un dato de tipo complejo, es decir, de la forma $a + bi$. CUDA implementa un índice para los números complejos, de la misma manera que para los hilos. Estos índices se representan con ".x" como la parte real a, ".y" como la parte imaginaria b. Obtenemos entonces, una salida de tamaño 1024 (la cantidad de datos en cada secuencia) por N (la cantidad de secuencias), donde cada dato es una potencia del dato original. Computamos entonces la ecuación de la potencia:

$$P_{xx}(f) = \int_{-\infty}^{\infty} |X(\omega)|^2 \, d\omega. \tag{6.1}$$

Código 6.1: Código de ejecución de la función para la potencia espectral. Esta función se ejecuta en el *Device*.

```
1  __global__ void Power(cufftComplex *Input, cufftComplex *Output, int nlen, int
       npts)
2  {
3      int ThreadPerBlock = blockDim.x*blockDim.y*blockDim.z;
4      int ThreadNumInBlock = (threadIdx.x + threadIdx.y*blockDim.x) + threadIdx.
           z*(blockDim.x*blockDim.y);
5      int BlockNumInGrid = (blockIdx.x + blockIdx.y*gridDim.x) + blockIdx.z*(
           gridDim.x*gridDim.y);
6
7      int globalThreadNum = ThreadNumInBlock + ThreadPerBlock*BlockNumInGrid;
8
9      if(globalThreadNum < nlen){
10         Output[globalThreadNum].x = (Input[globalThreadNum].x * Input[
               globalThreadNum].x + Input[globalThreadNum].y * Input[
               globalThreadNum].y)/npts;
11         Output[globalThreadNum].y = 0;
12     }
13 }
14
15 dim3 DimGrid(grid_size, grid_size, grid_size);
16 dim3 DimBlock(block_size, block_size, block_size);
17 Power<<<DimGrid, DimBlock>>>(Output_fft, Power_Out, size_fft*batch, size_fft);
```

Podemos referirnos a los hilos de diferentes maneras, su identificador por bloque o su identificador general dentro de la malla (ver Tabla 6.2), así podemos elegir que hilo de qué bloque queremos ejecutar. Es importante recordar que la ejecución de los hilos es de manera paralela y en grupos de 32 llamados warps. El orden de lanzamiento es indefinido. La cantidad de bloques que se ejecutarán en paralelo está determinado por el rendimiento de la GPU.

6.4. Implementación en GPU de la correlación

La correlación entre señales describe la semejanza que existe entre ambas. El segundo *kernel* que definimos entonces, es para calcular la correlación entre todas las señales. Teniendo en cuenta que la correlación de la señal 5 y la señal 1 es la

misma que la correlación de 1 y 5, por ejemplo. El tamaño de las comparaciones estará definido por $\frac{N*(N-1)}{2}$, donde N representa el número de señales. Así para un ejemplo de 5 señales podemos ver el ejemplo de la Tabla 6.3

1°	1-2	2-3	3-4	4-5
2°	1-3	2-4	3-5	
3°	1-4	2-5		
4°	1-5			

Tabla 6.3: Ejemplo de cantidad total de comparaciones para una muestra de 5 secuencias. Cada celda representa los índices de las muestras con que se pueden comparar. La forma en la que se guardarán las secuencias en el vector de salida es en el orden de izquierda a derecha siguiendo cada número de iteración, para cualquier N, donde N representa la cantidad de secuencias.

La función de correlación no hace todo el proceso de la función computada:

$$\frac{(a_1 + bi_1)(a_2 + bi_2)}{N\sqrt{Power_1 * Power_2}} \tag{6.2}$$

sino que solo paraleliza la operación $(a_1 + bi_1)(a_2 + bi_2)$ que nos da un vector salida de tamaño de 1024*N, donde N es el número total de comparaciones.

Código 6.2: Código de ejecución para la función de correlación.

```
1  __global__ void Correlation(cufftComplex *Input, cufftComplex *Output, int N,
       int size, int begin)
2  {
3      int ThreadPerBlock = blockDim.x*blockDim.y*blockDim.z;
4      int index = threadIdx.x + ThreadPerBlock*blockIdx.x;
5
6      Output[ index+begin ].x = Input[ index ].x * Input[ index + N*size ].x +
           Input[ index ].y * Input[ index + batch_id*size ].y;
7      Output[ index+begin ].y = Input[ index ].x * Input[ index + size*size ].y
           - Input[ index ].y * Input[ index + size*size ].x;
8  }
9
10 int Begin = 0;
11 for(int it=1; it<N; floor++){
12     Correlation<<< N-it, size >>>(Output_fft, Correlation_Out, it, size, Begin
           );
13     Begin += size*(N-it);
14 }
15
16 int id_v1 = 0;
17 int id_v2 = 1;
18 int v_id = id_v2;
19 int B = 0;
20 int E = batch-1;
21
22 for(int i=0; i<batch; i++){
23     for(int j=B; j<E; j++){
24         corr = 2*max_corr[j]/(DATASIZE*sqrt(Out_Power[id_v1*DATASIZE]*
               Out_Power[id_v2*DATASIZE]));
25         id_v1 += 1;
26         id_v2 += 1;
27     }
28     id_v1 = 0;
29     id_v2 = v_id+1;
30     v_id += 1;
31     B += batch-(i+1);
32     E += batch-(i+2);
33 }
```

Hay que mencionar que cada hilo solo se ejecuta una vez por *kernel*, para nuestra función hay que operar usando cada secuencia (vector) N-1 veces. Definimos entonces la estructura para llamar al *kernel*, y esta llamada la haremos $N-1$ veces, por lo que en cada iteración se ejecutará como lo muestra la Tabla 6.3. Para el ejemplo

de la tabla, en la primera iteración se ejecutarán cuatro comparaciones, es decir, un total de 1024 hilos * N-1 secuencias, que es 1024*4 = 4′096 hilos, luego 1024*3 hilos, 1024*2 hilos y 1024*1 hilos. El índice de cada vector donde se guardará la salida también variará en cada iteración. Los valores en cada iteración se muestran en la Tabla 6.4.

iteración	Índice de inicio	Hilos ejecutados	Hilos totales
1	0	4096	4,096
2	4,096	3072	7,167
3	7,168	2048	9,215
4	9,216	1024	10,240

Tabla 6.4: Para una muestra de cinco secuencias en forma de arreglos con una longitud de 1024, en la primera iteración el índice del vector donde se guardarán los resultados irá desde 0 hasta 4095, es decir, se ejecutarán 4,096 hilos, $1024 * 4$, ver Tabla 6.3. En la siguiente iteración el índice del vector donde se guardaran iniciara en 4,096 hasta la cantidad de hilos ejecutados, 3072, es decir, la siguiente sección del vector donde se guardarán será en el rango de índice que va de 4096 hasta 7168, así para cada iteración. El rango de índices de cada iteración estarán definido por: $\sum_{N-1}^{N-i} 1024 * N$, donde N es el número de secuencias y donde $i = 1, ..., N$. La cantidad de hilos ejecutada en cada iteración sera $\sum_{i=N-1}^{i=1} 1024 * i$, es decir, en cada iteración el número de hilos ejecutados sera cada vez menor.

6.5. Implementación en GPU de la coherencia espectral

La coherencia es usada, al igual que la correlación para medir la semejanza entre dos señales, pero la coherencia generalmente calcula la transferencia de potencia de entrada y salida en un rango de frecuencia de las señales. La función de coherencia se implementó con los mismos argumentos que la función de correlación, por lo que cumplirá con los valores de la Tabla 6.4. Implementando la función:

$$C_{xy} = \frac{|P_{xy}^2|}{P_{xx}P_{xy}}$$

Donde P_{xx} y P_{yy} son estimaciones de la potencia espectral de X y Y, P_{xy} es la densidad espectral cruzada de X y Y, la densidad espectral cruzada es la Transformada de Fourier de la función de correlación cruzada.

Código 6.3: Código de ejecución para la función de coherencia

```
1  __global__ void Coherence( cufftComplex *Input, cufftComplex *Output, int
       batch_id, int size, int begin )
2  {
3      int ThreadPerBlock  = blockDim.x*blockDim.y*blockDim.z;
4      int index = threadIdx.x + blockIdx.x*ThreadPerBlock;
5
6      Output[ index+begin ].x =  Input[ index ].x * Input[ index+batch_id*size
           ].x + Input[ index ].x * Input[ index+batch_id*size ].y;
7      Output[ index+begin ].y =  Input[ index ].x * Input[ index+batch_id*size
           ].y + Input[ index ].y * Input[ index+batch_id*size ].x;
8  }
9
10 int Begin = 0;
11 for( int floor=1; floor<batch; floor++ ){
12         Coherence<<< batch-floor, size_fft >>>(Output_fft, Coherence_Out,
               floor, size_fft, Begin);
13         Begin += size_fft*(batch-floor);
14 }
15
16 int id_v1 = 0;
17 int id_v2 = 1;
18 int v_id = id_v2;
19 int B = 0;
20 int E = batch-1;
21
22 for( int i=0; i<batch; i++ ){
23     for( int j=B; j<E; j++ ){
24         coh = max_cohr[j]/(Out_Power[id_v1*DATASIZE]*Out_Power[id_v2*DATASIZE
               ]);
25         id_v1 += 1;
26         id_v2 += 1;
27     }
28     id_v1 = 0;
29     id_v2 = v_id+1;
30     v_id += 1;
31     B += batch-(i+1);
32     E += batch-(i+2);
33 }
```

6.6. Transferencia de datos GPU/CPU

Tenemos, entonces, tres vectores de una dimensión como salida, la potencia espectral, la coherencia y la correlación. Para operar sobre estos vectores fuera del *Device*, tenemos que usar las misma funciones de `cudamemcpy` para operar en el *Host*. Antes de mover los datos del *Device* al *Host*, transformamos el tipo de dato de complejo a real que es el tipo de dato original de la secuencia, mediante la función `cufftExecC2R`. Una vez copiados los datos al *Host*, podemos ver directamente las potencias que se encontrarán dentro del vector potencia. Cada valor inicial de las secuencias en el vector será el valor de la potencia espectral, es decir, $i * 1024$ donde i es el índice de secuencia.

Para determinar la correlación, aplicamos el cálculo del máximo de cada secuencia dentro del vector resultante de correlación. Siguiendo la función expuesta en la sección 5.4, la Ecuación 5.48, el valor máximo de cada vector será dividido entre la raíz de la multiplicación de las potencias espectrales de las secuencias que representan en el vector de correlación. En el ejemplo de la Tabla 6.3, vemos las comparaciones posibles usando los índices de las secuencias. Si operamos con la correlación de las secuencias 1 y 3, que estará guardada en la posición del vector $4*1024$ (por lo explicado en la tabla), y por lo que tendríamos que usar las potencias de la secuencia 1 y la secuencia 3, extraídas directamente del vector salida de la potencia espectral. Para el último procesamiento de la coherencia, deben seguirse los mismos pasos, pero sin operar con la raíz cuadrada en las potencias.

CAPÍTULO 7

DISCUSIÓN Y RESULTADOS

El uso de tarjetas gráficas de propósito general para el procesamiento de grandes cantidades de datos, permitió el análisis de un volumen importante de señales sísmicas en un tiempo muy reducido en comparación con un procesado de datos serial en CPU. Se logró una reducción en el tiempo de procesado de los datos hasta 75 veces menor en comparación con el procesamiento en CPU, en una prueba de rendimiento se procesaron 10,000 archivos con señales sísmicas, cada archivo representa un sismo reportado por el Servicio Sismológico Nacional con 1,024 datos de tipo flotante. Haciendo una comparación contra todo el conjunto con un total de 49,995,000 de comparaciones de secuencias y 51,194,880,000 datos totales por procesar.

El uso de la GPU permite el aprovechamiento de miles de núcleos de proceso (cores) que al estar construido físicamente en la misma unidad aritmética lógica facilita no sólo la transferencia de datos, sino que también permite una fácil paralelización de los procesos y puede utilizarse en diversas tarjetas gráficas con una mínima modificación del código. La GPU mediante funciones CUDA, usa el poder del procesador y aplica operaciones matemáticas que luego serán copiadas a la CPU para obtener los resultados.

Parte del procesamiento de los registros sísmicos, se basó en el análisis espectral usando la librería cuFFT desarrollada por NVIDIA. Con base en las funciones para el cálculo de la Transformada de Fourier, se crearon nuevas funciones para el cálculo de la correlación y la coherencia espectral. Se compararon los tiempos de ejecución en diferentes GPU de gama distinta, una NVIDIA Geforce 745m perteneciente a generaciones atrasadas contra una NVIDIA Geforce 1070 pertenecientes a gamma alta

de última generación (2016). La Tabla 7.1 lista los tiempos de ejecución de ambas tarjetas para los procesos más importantes.

Proceso	Tiempo en Geforce 745m	Tiempo en Geforce 1070
cuDeviceTotalMem	14.60us	20.47us
cudaLaunch	32.562us	34.672us
cudaFree	5.7840us	2.4960us
cudaMalloc	5.4610us	2.4830us
cudaGetDevice	4.5920us	1.8160us
cudaMemcpy	2.2660us	1.1910us

Tabla 7.1: Comparación de tiempos de ejecución en tarjetas NVIDIA

En la Figura 7.1, se muestra una comparación de los tiempos de ejecución entre el código de correlaciones ejecutado de manera secuencial en CPU usando el lenguaje Python 3, un Intel core i7 y su código equivalente ejecutado en la GPU (CUDA) con una tarjeta Geforce GTX 1070. Se utilizaron cuatro conjuntos de archivos de prueba de 10, 100, 1000 y 10,000 archivos de secuencias sismicas. Por ejemplo, para el conjunto de 10,000 archivos, el procesamiento en el CPU tomó 3 horas con 49 minutos, mientras que el tiempo de ejecución en la GPU fue de únicamente 2 minutos con 53 segundos lo que representa un tiempo de ejecución 75 veces menor. Esto permite la ejecución rápida de miles de archivos, además se observa que la diferencia en los tiempos de ejecución se vuelve cada vez mayor a medida que aumenta el volumen de archivos a procesar.

Los resultados del algoritmo se muestran en la Tabla 7.2. La correlación de la señal consigo misma es 1, al igual que la coherencia. Los tiempos de ejecución son muy bajos al tratarse de la ejecución de una sola señal.

Para una señal cualquiera, ejecutamos de la Transformada de Fourier en CPU y en GPU. En la Figura 7.2, podemos ver la salida gráfica para una señal de entrada aplicando la Transformada de Fourier. Los picos de la señal determinan las ondas

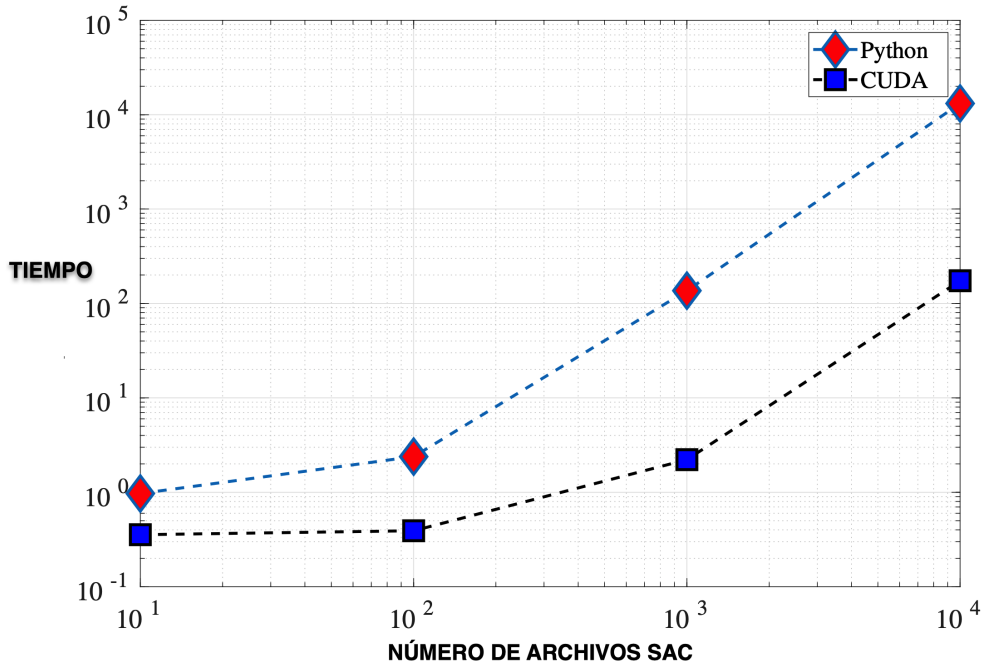

Figura 7.1: Comparación de tiempos de ejecución para diferentes conjuntos de secuencias ejecutados con CUDA en la GPU NVIDIA Geforce GTX 1070 y Python en una CPU Intel core i7 - 4.00 HGz.

de la señal original en el espacio bidimensional para cada valor en la frecuencia. En la Figura 7.3 se muestran los resultados gráficos de la correlación para dos señales, para la cual, una de ellas es la señal usada en la Figura 7.2.

7.1. Validación del código

Para la validación del código implementado, se realizaron pruebas usando la función seno desfasadas en el tiempo, de la forma $y(t) = sin(2\pi ft - \phi)$. Se generaron un total de 13 archivos ".sac", cada uno con un desplazamiento de 30 grados entre sí, iniciando de 0 hasta llegar a los 360. Las funciones seno se computaron en el

Device name	GPU	CPU
Coherence	1.00	1.00
Correlation	1.00	1.00
Memory Clock Rate (KHz)	600000	9000000
Memory Bus Width (bits)	128	50
Peak Memory Bandwidth (GB/s):	28.800000	2.600000

Tabla 7.2: Salida de ejecución del algoritmo usando la función seno como entrada.

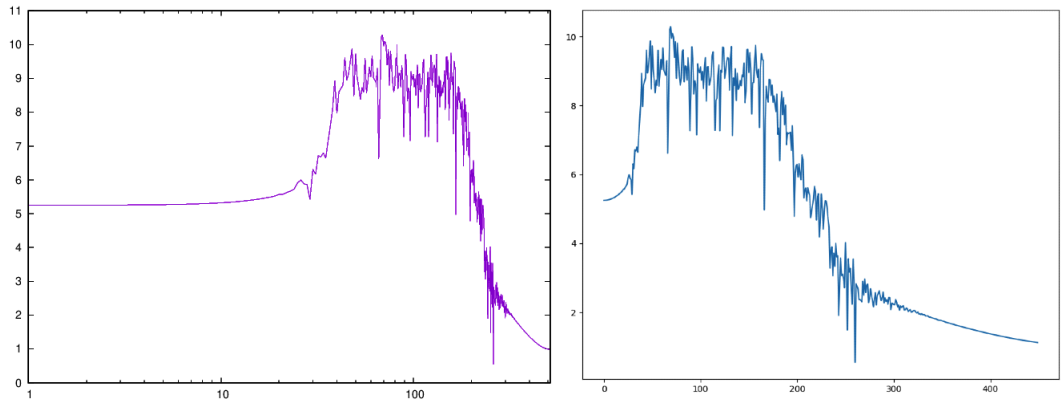

Figura 7.2: En la gráfica de la derecha, vemos la salida para la ejecución de una señal en CPU usando Python, en la gráfica de la izquierda se muestra la salida de la ejecución en GPU usando CUDA.

software SAC usando el Código 7.1 modificando el argumento de inicio con los grados deseados. Una visualización gráfica se muestra en la Figura 7.4. La función seno generada se inicializó con 25 segundos de duración, un periodo de $2.92\,s$ y una frecuencia de $f = 0.255\,Hz$. Todas las señales generadas tienen una frecuencia de muestreo $10Hz$ y un total de 256 puntos. Los puntos en color rojo indican cual es el desfase necesario para lograr que la correlación sea máxima.

Código 7.1: Los argumentos de la función están definidos de la siguiente manera: FUNCGEN TYPE, DELTA v, NPTS n, BEGIN v. Donde *TYPE* es definido como *SINE v1 v2*, una función seno con la frecuencia en Hz dada por el ángulo v1 y la fase en grados por v2. *Delta*, como el conjunto de incremento mínimo entre las muestras

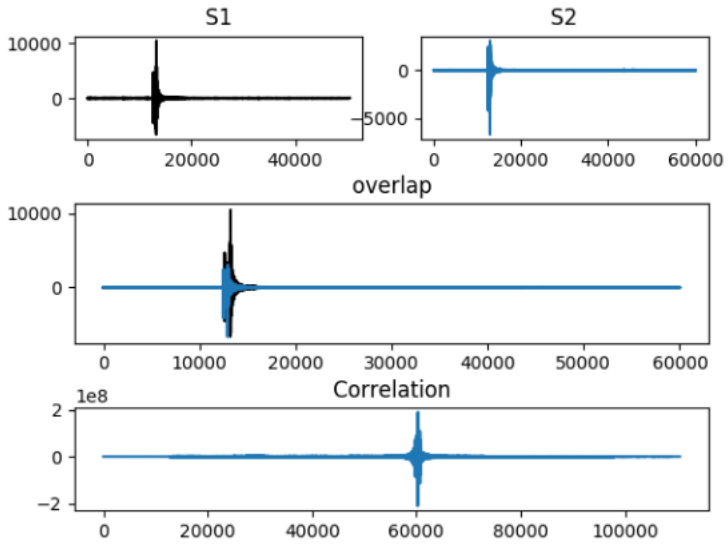

Figura 7.3: Ejemplo gráfico de la correlación de dos señales.

a v. *NPTS*, el número de puntos en la secuencia. *BEGIN*, el momento de inicio de la función.

```
1 SAC> funcgen delta 0.1 npts 256 sin 0.255 0.0
2 SAC> write seno000.sac
```

Con el fin de reducir el fenómeno de Gibbs [Kelly, 1996], el cual introduce oscilaciones de alta frecuencia al cálculo de la FFT, cuando existen discontinuidades en la señal, se aplicó un *taper* cosenoidal al 5 % en ambos extremos utilizando la función de SAC, *taper*, ver Figura 7.4, el *taper* es aplicado ya que, cuando se realiza la FFT, asumimos que la señal es una señal periódica, es decir los valores en los extremos deben de ser iguales en esta Figura además se muestra la corrección por desfase de la señal, de acuerdo al valor obtenido a partir de la variable *ind_max_corr*.

Esta variable se calcula a partir de encontrar el valor que produce el máximo coeficiente de correlación. Comprobamos entonces, la precisión del análisis mediante las salidas de correlación de las funciones sinusoidales. Podemos ver que la correlación, en algunos casos toma un valor de 1.002 y 1.008 lo cual representa un error

del 0.2 % y 0.8 % con respecto al valor esperado de 1.0. Podemos atribuir estas desviaciones del valor esperado a errores de redondeo ocurrido durante el cálculo del coeficiente de correlación. La Figura 7.5 muestra la variación en el coeficiente de correlación en función del desfase de la señal. El panel superior de esta figura muestra todas las señales superpuestas después de haber hecho la corrección por desfase. Como se observa, la variación en el coeficiente de correlación se puede atribuir al uso del *taper* en los extremos de la señal, lo cual ocasiona una disminución del $\tilde{1}0\,\%$ en el cálculo del coeficiente de correlación.

7.2. Efecto del ruido

Un aspecto importante que nos interesa conocer es el efecto que tiene el ruido sobre el cálculo del coeficiente de correlación. La clasificación de un evento, como sismo repetitivo o no depende principalmente de que el coeficiente de correlación sea superior al 95 %. Sin embargo, el ruido puede afectar a que este coeficiente disminuya debido a la presencia de agentes externos como son: ruido urbano el cual es variable dependiendo de la hora del día, la presencia del viento y lluvia, variaciones en el oleaje (esto afecta principalmente a estaciones cercanas a la costa), etc. Con el fin de evaluar cuál es el efecto producido cuando se añade ruido blanco a la señal, se generaron un total de 30 señales sinusoidales a las cuales se les añadió ruido proporcional a la amplitud máxima de la señal. Las señales contaminadas se muestran en la Figura 7.6.

A cada señal se le agregó un porcentaje de ruido blanco que va del 1 % (señal superior) hasta el 30 %. En todos los casos se aplicó un *taper* al 5 % en ambos extremos de la señal y se obtuvo el coeficiente de correlación tomando como base la señal superior. La Figura 7.7 muestra la variación del coeficiente de correlación en función del porcentaje de ruido agregado, así como las señales utilizadas superpuestas. Se observa que el coeficiente de correlación disminuye conforme se aumenta el ruido a la señal hasta alcanzar un mínimo de alrededor del $\tilde{1}0\,\%$ menos del valor esperado

en el caso de 30 % de nivel de ruido. De igual forma que en la prueba anterior, se observa que existe una sobre-estimación del valor esperado del 0.8 % en el caso de la auto-correlación de la señal.

En general, no existe control sobre el nivel de ruido de la señal, y no se esperan que las variaciones del ruido de las estaciones permanentes del servicio sismológico nacional tengan incremento tan altos en el nivel de ruido debido a la estructura que alberga dichas estaciones. Por lo que podemos considerar que pueden existir variaciones producidas por el ruido de alrededor del ± 5 % en casos extremos.

7.3. Detección de eventos repetitivos

El procesado de las señales arrojó un conjunto de 1084 eventos repetitivos. Estos eventos repetitivos contienen al menos 2 sismos cuyos coeficientes de correlación y coherencia superan el ≥ 0.95. La secuencia con más miembros contiene 15 sismos ocurridos entre el 2012 y 2018, con una magnitud promedio de M 3.6. La Figura 7.8 muestra la localización de estos grupos de eventos a lo largo de la costa y su ocurrencia en el tiempo. En esta figura se observa que existe un incremento notable en la actividad de sismos repetitivos después de los sismos del 20 de marzo del 2012, M 7.4 en Ometepec, Guerrero y del sismo del 18 de abril del 2014, Mw 7.2 en Tecpan Guerrero. Lo que permite analizar los cambios abruptos en el desplazamiento post sísmico debido a estos dos terremotos. Cabe señalar que las zonas correspondientes a la zona de ruptura del sismo del 19 de septiembre de 1985 y un segmento de la brecha de Guerrero no presentan la existencia de sismos repetitivos. Ante esta situación podemos sugerir dos casos hipotéticos que justifican la ausencia de sismos repetitivos en esta zona: (1) las placas se encuentran completamente acopladas, es decir, no existe movimiento relativo entre estas y se encuentran en un estado de acumulación de esfuerzos; (2) estas zonas se encuentran completamente desacopladas, es decir, el desplazamiento entre placas se da de forma asísmica debido a la falta de asperezas. Para el caso de la zona de ruptura del sismo de 1985, la primera hipótesis es más

factible y es consistente con las altas tasas de deformación encontradas en esta área. Por otra parte, para la zona de la brecha de Guerrero, la segunda hipótesis es más factible debido a la presencia de sismos lentos documentados en la zona.

Las tasas de deslizamiento se obtuvieron a partir de la relación [Nadeau and Johnson, 1998]. La Figura 7.9 muestra el cálculo de la tasas de deslizamiento a partir de los sismos repetitivos encontrados para un instante de tiempo previo al sismo del 20 de marzo del 2012, en Ometepec, Guerrero de acuerdo al procedimiento descrito en la sección 4.4. En este mapa se observan las variaciones en las zonas donde la tasa de deslizamiento es mayor a consecuencia del desplazamiento ocurrido previo al sismo. Los círculos indican la sismicidad previa de magnitud $\geq M5$, el color de cada cuadro indica el desplazamiento en centímetros por año. Mientras que las flechas indican las tasas de subducción geodésicas [DeMets et al., 2010]. Este mapa permite el análisis de las variaciones espacio-temporales del desplazamiento y observar en qué zonas ocurrió un mayor desplazamiento y cómo se relaciona con la sismicidad. En este sentido, se observa que las zonas con mayor actividad sísmica, en particular, la región que comprende Ometepec, Guerrero y Pinotepa Nacional, Oaxaca es la zona que históricamente registra la mayor actividad sísmica y que a su vez registran un constante deslizamiento debido a sismos lentos con tasa de recurrencia que van de los 1-2 años, y magnitudes equivalentes a un sismo de magnitud M6.

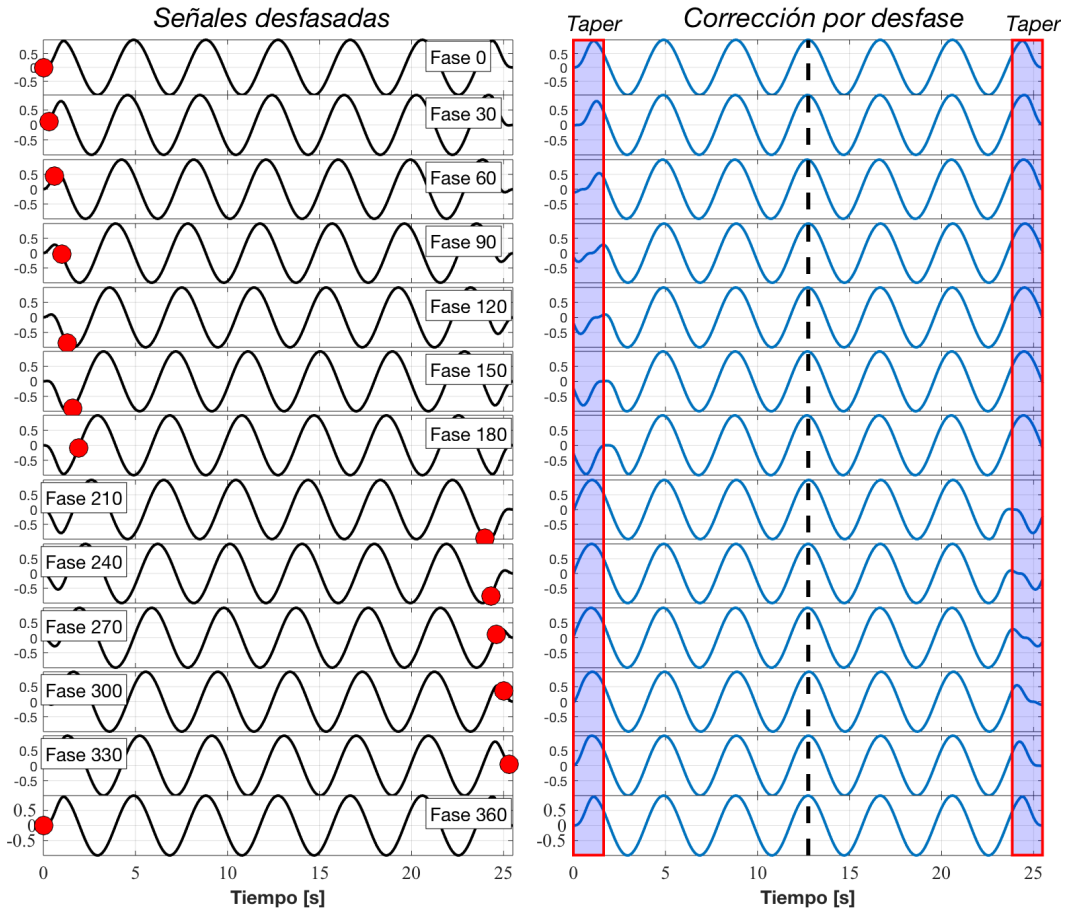

Figura 7.4: (Izquierda) Funciones senos sintéticas de prueba de la forma $y(t) = \sin(\pi\, ft - \phi)$. Cada señal tiene un desfase de 30 grados entre sí. Los puntos en color rojo indican el corrimiento necesario para producir la señal con máximo coeficiente de correlación. (Derecha) Ajuste por medio del corrimiento de los datos necesario para producir una máxima correlación. El área sombreada indica la zona donde se aplicó la función *taper* [2] a a señal, la línea punteada muestra que las señales se encuentran en fase.

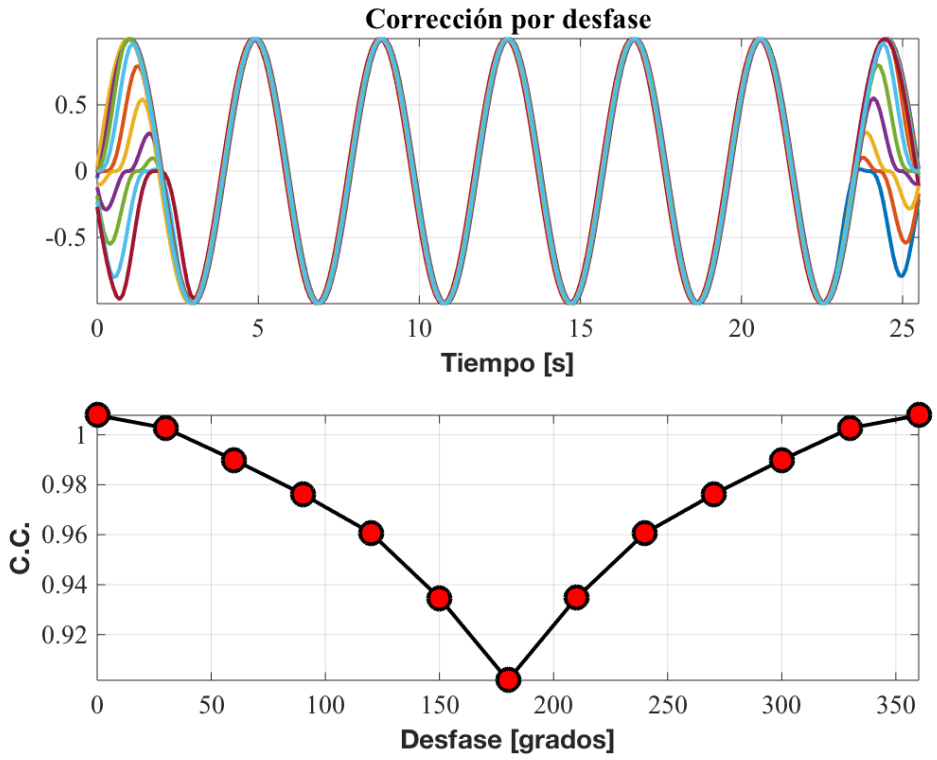

Figura 7.5: La gráfica superior muestra la corrección de desfase de cada función seno, mientras que en la gráfica inferior muestra como el grado de correlación disminuye cuando el desfase entre las señales se incrementa.

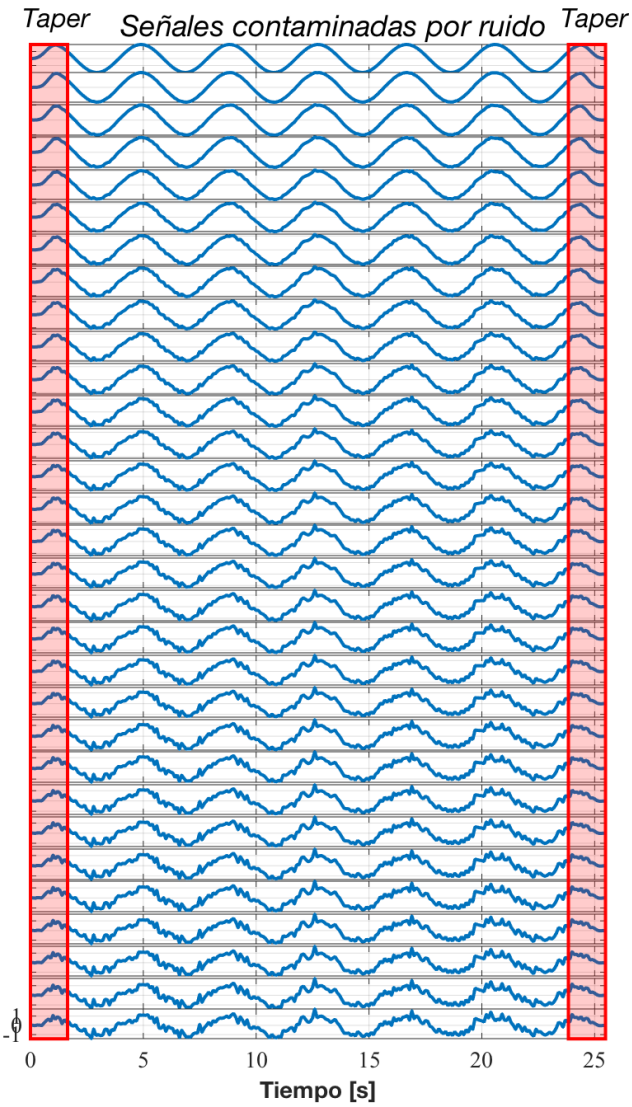

Figura 7.6: Conjunto de señales sinusoidales contaminadas por ruido Gaussiano. Se añadió de manera porcentual con incrementos del 1 % de la amplitud máxima de la señal original desde el 1 % señal hasta el 30 %. Además, se aplicó un *taper* al 5 % en ambos extremos.

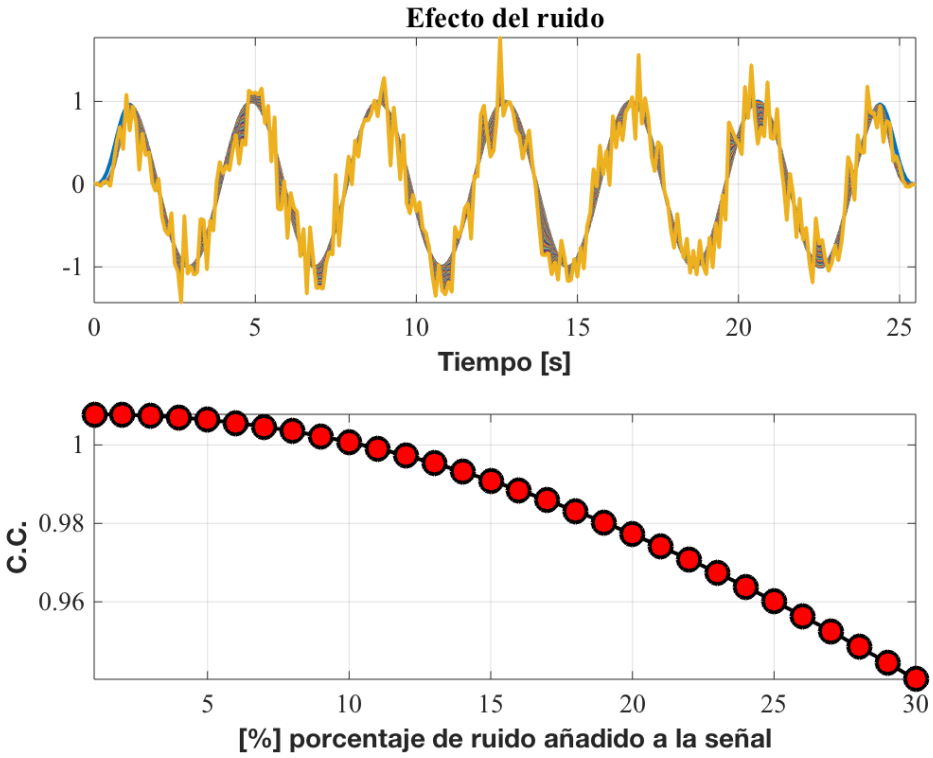

Figura 7.7: Efecto del ruido sobre el cálculo del coeficiente de correlación. (Arriba) superposición de señales con ruido añadido, todas las señales se encuentran en fase. (Inferior) Variación del coeficiente de correlación en función del ruido añadido.

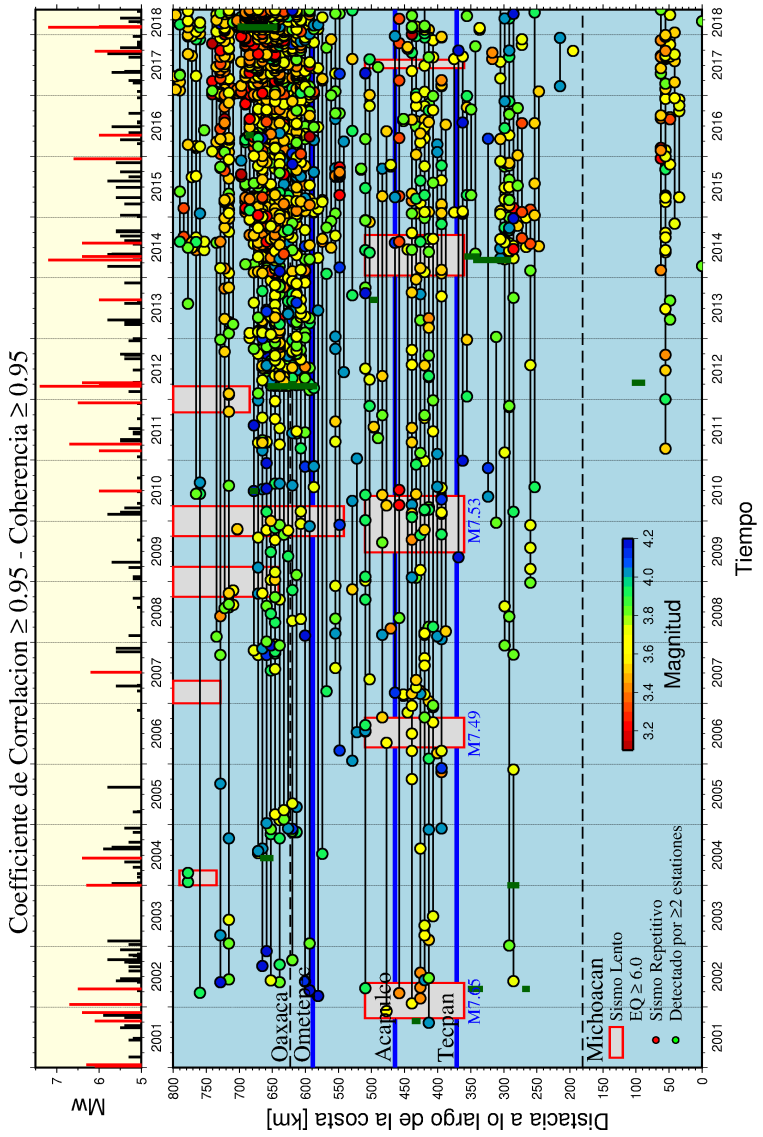

Figura 7.8: Relación espacio temporal de ocurrencia de sismos repetitivos a lo largo de la costa del Pacífico. El panel superior indica la actividad sísmica reportada por el SSN para sismos mayores de $\geq M5.0$, el panel inferior indica la presencia de sismos repetitivos de acuerdo con su localización a lo largo de la trinchera. La zonas sombreadas indican la ocurrencia de sismos lentos en la zona en las brechas sísmicas de Guerrero y Oaxaca. El color indica la magnitud promedio.

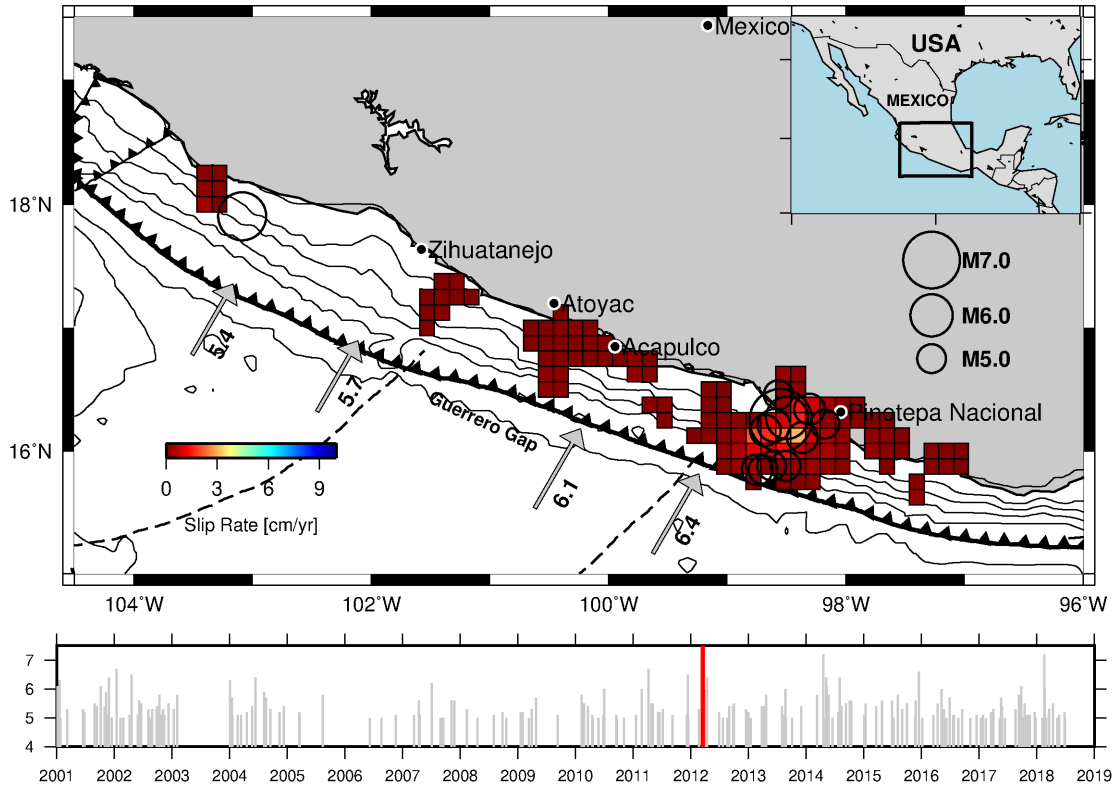

Figura 7.9: Análisis de las tasas de deslizamiento, inmediatamente posteriores al sismo del 20 de marzo, 2012 en Ometepec, Guerrero. Los círculos indican la sismicidad en las dos semanas previas al sismo para sismos mayores a M5.0 (panel inferior). La cuadricula está coloreada de acuerdo a la tasa de desplazamiento inferida a partir de los sismos repetitivos encontrados.

CAPÍTULO 8

CONCLUSIONES

El procesamiento utilizando una tarjeta GPU NVIDIA GeForce GTX 1070 demostró ser una herramienta poderosa con múltiples ventajas. Al estar integrado todas las unidades de proceso en una sola unidad, permitió que los tiempos de transferencia de datos al interior del equipo de cómputo fueran muy cortos en comparación con otras técnicas de procesamiento de multiprocesadores, o de cómputo distribuido, las cuales pueden llegar a generar grandes tiempos de transferencia de datos. En la Tabla A.1 y la Tabla A.2 se muestran los resultados de las pruebas de rendimiento del código con 10 mil señales de entrada. Las tablas describen el tiempo de ejecución total para el procesamiento de cada operación en el kernel. La columna Min/Max/AVG muestran los tiempos de ejecución del proceso, mientras que Min y Max miden el tiempo mínimo y máximo de los tiempos por ejecución, AVG mide el promedio de todas las ejecuciones, si el proceso se ejecutó una vez los tres valores deberían ser iguales.

El tiempo de procesamiento de la GPU NVIDIA GeForce GTX 1070 logró tener un tiempo de ejecución 6 veces menor que la tarjeta NVIDIA 745m, hay que notar que se usaron dos modelos de CPU. En el computador 1, una CPU Intel Core i7 - 4.00 GHz con una GPU NVIDIA Geforce 1070, y en el computador 2 con una CPU Intel Core i7 - 1.80 GHz con una GPU Geforce 745m.

Los resultados muestran que si bien el tiempo de ejecución entre la NVIDIA Geforce 1070 y su respectivo CPU, en la Figura 7.1; tienen una diferencia exponencial, estos resultados se podrían comparar contra los resultados de tiempos de ejecución

de la otra GPU usada, cuyos resultados se muestran en la Tabla 7.1. Habría que aludir a ley de Amdahl para entender porque una CPU del mismo modelo en diferentes sistemas puede tener un mejor desempeño que la GPU, y esto es porque los componentes de su sistema son mas rápidos que los componentes del GPU 745m. Los resultados muestran una disminución esperada en el tiempo de ejecución para el procesamiento de los datos, lo cual permite la detección de sismos repetitivos en muy pocos minutos, a diferencia del tiempo de ejecución obtenido por métodos convencionales en CPU.

Lo anterior es sumamente importante debido a que este tipo de análisis requiere del procesamiento de un volumen cada vez mayor de datos que crece día a día, además de que abre la posibilidad de incorporar datos de otras regiones del mundo. Para este estudio, se utilizaron estaciones sísmicas permanentes pertenecientes al Servicio Sismológico Nacional, cabe resaltar que esta institución ha tenido un crecimiento importante en su capacidad de monitoreo en la última década con la instalación de número mayor de estaciones sísmicas permanentes en el sureste y norte de México. Esto asociado a la ocurrencia de sismos con magnitudes $M > 7.0$ en los últimos cinco años, ha incrementado significativamente el número de datos a procesar.

8.1. Trabajo a futuro

Las tarjetas gráficas son dispositivos de uso interno que han evolucionado de forma rápida en los últimos años. En periodos muy cortos de tiempo, de tan solo unos cuantos meses, aparecen nuevos modelos de tarjetas gráficas, cada vez con un número mayor de núcleos y memoria interna que hace que las tarjetas anteriores se vuelvan rápidamente obsoletas. Por lo que para mantener un equipo de cómputo actualizado, se requiere de una continua inversión económica y la rápida sustitución de los elementos físicos dentro del equipo de cómputo.

Por este motivo, se sugiere, en un trabajo futuro, realizar el procesamiento de los datos a través de un servicio de cómputo en la nube. Esto no solo reduce costos, sino que permite utilizar nuevos modelos de tarjetas gráficas sin la necesidad de instalarlos físicamente. Algunos servicios populares al momento de la conclusión de esta tesis son Amazon Web Services (AWS) y Google Cloud. Estos proveedores de cómputo en la nube ofrecen la posibilidad de realizar el procesamiento de los datos pagando únicamente por el tiempo y los recursos necesarios, sin tener la necesidad de comprar físicamente un equipo, con el costo de mantenimiento que esto implica.

APÉNDICE A

Tablas de rendimiento

Profiling result GPU NVIDIA Geforce 745m							
Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU	activities: 100.00 %	8.6392ms	1	8.6392ms	8.6392ms	8.6392ms	VectorMult
API calls:	62.02 %	74.118ms	2	37.059ms	3.4050us	74.115ms	cudaEventCreate
	28.90 %	34.534ms	1	34.534ms	34.534ms	34.534ms	cudaDeviceReset
	7.40 %	8.8459ms	1	8.8459ms	8.8459ms	8.8459ms	cudaEventSynchronize
	0.95 %	1.1314ms	185	6.1150us	395ns	267.31us	cuDeviceGetAttribute
	0.41 %	494.15us	1	494.15us	494.15us	494.15us	cudaGetDeviceProperties
	0.13 %	151.60us	2	75.799us	68.020us	83.579us	cuDeviceTotalMem
	0.12 %	145.59us	2	72.797us	63.189us	82.405us	cuDeviceGetName
	0.03 %	32.562us	1	32.562us	32.562us	32.562us	cudaLaunch
	0.01 %	16.588us	2	8.2940us	7.2650us	9.3230us	cudaEventRecord
	0.01 %	6.5320us	1	6.5320us	6.5320us	6.5320us	cudaDeviceSynchronize
	0.00 %	5.7840us	4	1.4460us	1.2460us	1.9430us	cudaFree
	0.00 %	5.4610us	6	910ns	497ns	1.5610us	cudaMalloc
	0.00 %	4.6900us	2	2.3450us	1.9010us	2.7890us	cudaEventDestroy
	0.00 %	4.5920us	3	1.5300us	1.0640us	2.2650us	cudaGetDevice
	0.00 %	3.8930us	4	973ns	399ns	2.0510us	cuDeviceGetCount
	0.00 %	3.2630us	1	3.2630us	3.2630us	3.2630us	cudaEventElapsedTime
	0.00 %	2.4340us	3	811ns	491ns	1.1970us	cuDeviceGet
	0.00 %	2.2660us	3	755ns	489ns	1.0340us	cudaMemcpy
	0.00 %	1.3460us	1	1.3460us	1.3460us	1.3460us	cuInit
	0.00 %	1.2200us	4	305ns	192ns	481ns	cudaSetupArgument
	0.00 %	1.0000us	1	1.0000us	1.0000us	1.0000us	cudaConfigureCall
	0.00 %	573ns	1	573ns	573ns	573ns	cuDriverGetVersion
	0.00 %	325ns	1	325ns	325ns	325ns	cudaGetDeviceCount

Tabla A.1: Prueba de estabilidad en GPU Nvidia GeForce GTX 1070.

Profiling result NVIDIA 1080							
Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU	activities: 100.00 %	1.1309ms	1	1.1309mss	1.1309ms	1.1309ms	VectorMult
API calls:	71.27 %	114.45ms	2	57.223ms	843ns	114.45ms	cudaEventCreate
	27.75 %	44.567ms	1	44.567ms	44.567ms	44.567ms	cudaDeviceReset
	0.35 %	561.70us	1	561.70us	561.70us	561.70us	cudaEventSynchronize
	0.29 %	467.88us	169	2.7680us	104ns	106.55us	cuDeviceGetAttribute
	0.14 %	228.70us	1	228.70us	228.70us	228.70us	cudaGetDeviceProperties
	0.12 %	200.47us	2	100.24us	97.098us	103.38us	cuDeviceTotalMem
	0.04 %	56.576us	2	28.288us	27.383us	29.193us	cuDeviceGetName
	0.02 %	34.672us	1	34.672us	34.672us	34.672us	cudaLaunch
	0.00 %	6.8090us	2	3.4040us	2.8640us	3.9450us	cudaEventRecord
	0.00 %	3.6910us	1	3.6910us	3.6910us	3.6910us	cudaDeviceSynchronize
	0.00 %	2.4960us	3	832ns	737ns	985ns	cudaGetDevice
	0.00 %	2.4830us	1	2.4830us	2.4830us	2.4830us	cudaEventElapsedTime
	0.00 %	1.9030us	4	475ns	139ns	1.3650us	cudaSetupArgument
	0.00 %	1.8160us	6	302ns	130ns	652ns	cudaMalloc
	0.00 %	1.4940us	2	747ns	404ns	1.0900us	cudaEventDestroy
	0.00 %	1.4680us	4	367ns	96ns	1.0190us	cuDeviceGetCount
	0.00 %	1.3270us	3	442ns	140ns	737ns	cudaMemcpy
	0.00 %	1.1910us	3	397ns	139ns	782ns	cuDeviceGet
	0.00 %	798ns	1	798ns	798ns	798ns	cudaConfigureCall
	0.00 %	746ns	1	746ns	746ns	746ns	cuInit
	0.00 %	581ns	4	145ns	123ns	209ns	cudaFree
	0.00 %	515ns	1	515ns	515ns	515ns	cuDriverGetVersion

Tabla A.2: Prueba de estabilidad en GPU NVIDIA 1080.

Glosario

bloques Conjunto de hilos que se ejecutaran y cooperaran entre sí. Cada bloque agrupa los hilos en conjuntos de 32 hilos para su ejecución. 28

hilos Registro de un proceso que se ejecutará en cierto orden. 28

IEEE (*Institute of Electrical and Electronics Engineers*) Organización de desarrollo tecnológico que se dedica a la estandarización ´y desarrollo de áreas técnicas. 12

malla Conjunto de bloques que pueden o no ser ejecutados independientemente. 28

memoria DRAM "Dynamic RAM", tiene la característica de eliminar la información en la memoria cuando no tiene una fuente de energía y necesita recargar sus capacitores cada cantidad de tiempo para no perder los datos. Suele ser más barata, pero mas lenta. 17

memoria RAM "Random Access Memory", memoria que almacena datos que están en uso, facilitando el rápido acceso a la información solicitada por el procesador. 18

memoria SRAM "Static Ram", también conocida como memoria caché. Se encarga de mantener los datos con mayor uso en la memoria permitiendo así una mayor velocidad sin necesidad de recargar sus capacitores, aunque al aislar de una fuente de energía, puede perder la información si es de tipo volátil o no. 17

memoria VRAM "Video RAM", es la memoria que se encarga de gestionar los datos relevantes con el video o la gráfica de la computadora, es exclusiva para la GPU, como la RAM a la CPU. 18

primitivas Agrupación de vértices que dan como resultado una figura geométrica, desde un punto hasta un polígono. 18

Stream Processors (SP) Son los núcleos de ejecución de los cuales esta conformado una GPU y se encargan de realizar las operaciones. En algunas fuentes pueden referirse a ellos como CUDA cores, aunque hay quienes los diferencian dependiendo de la arquitectura en la que se emplee. 17

Streaming Multiprocessor (SM) Procesadores encargados de ejecutar los códigos CUDA. Gestiona el proceso de ejecución de los hilos y realiza las operaciones que están programadas para cada porción del código mediante los stream processors. 17

warp Conjunto de 32 hilos que ejecutaran el mismo código al mismo tiempo. Puede que los subprocesos de cada hilo se comporte diferente a pesar de que se ejecutan al mismo tiempo y procesan el mismo código. 23

Bibliografía

[Cano, 2018] Cano, A. (2018). A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8:e1232.

[Carreon, 2016] Carreon, N. R. (2016). *Ambientes virtuales con GPU y sensores RGB-D*. PhD thesis, Universidad Nacional Autónoma de México.

[Casas, 2016] Casas, J. G. (2016). Sistemas numéricos y representación de números en una computadora.

[Cobo, 2011] Cobo, J. O. (2011). Computación heterogénea.

[Cox, 1970] Cox, D. R. (1970). *The analysis of binary data [by] D. R. Cox*. Methuen London.

[César Represa Pérez, 2016] César Represa Pérez, José María Cámara Nebreda, P. L. S. O. (2016). Introducción a la programación en CUDA. Departamento de Ingeniería Electromecánica. Universidad de Burgos.

[DeMets et al., 2010] DeMets, C., Gordon, R. G., and Argus, D. F. (2010). Geologically current plate motions. *Geophysical Journal International*, 181(1):1–80.

[Dominguez et al., 2016] Dominguez, L. A., Taira, T., and Santoyo, M. A. (2016). Spatiotemporal variations of characteristic repeating earthquake sequences along the middle america trench in Mexico. *Journal of Geophysical Research: Solid Earth*, pages 3–7, 11–13.

[Dr. Vladimir Kostoglodov, 1999] Dr. Vladimir Kostoglodov, y. D. J. F. P. (1999). Cien años de sismicidad en México.

[Díaz, 2014] Díaz, J. M. G. (2014). Uso correcto de la correlación cruzada en climatología: el caso de la presión atmosférica entre taití y darwin. XXX(47):79–100.

[Encyclopedia, 2019] Encyclopedia, B. O. (2019). Computer.

[Francesc Guim, 2014] Francesc Guim, I. R. (2014). Best practices for scientific computing.

[Gebremedhin et al., 2012] Gebremedhin, M., Hemmati Moghadam, A., Fritzson, P., and Stavåker, K. (2012). A data-parallel algorithmic modelica extension for efficient execution on multi-core platforms.

[Geller and Mueller, 1980] Geller, R. J. and Mueller, C. S. (1980). Four similar earthquakes in central California. *Geophysical Research Letters*.

[Hanks and Kanamori, 1979] Hanks, T. C. and Kanamori, H. (1979). A moment magnitude scale. *Journal of Geophysical Research: Solid Earth*, 84(B5):2348–2350.

[Hayes, 1998] Hayes, M. H. (1998). *Schaum's Outline of Digital Signal Processing*. McGraw-Hill, Inc., New York, NY, USA, 1st edition.

[Hyyrö et al., 2005] Hyyrö, H., Fredriksson, K., and Navarro, G. (2005). Increased bit-parallelism for approximate and multiple string matching. *Journal of Experimental Algorithmics (JEA)*, 10:1–3.

[ILLESCA, 2005] ILLESCA, P. L. (2005). Transformada rápida de Fourier FFT.

[Kelly, 1996] Kelly, S. E. (1996). Gibbs phenomenon for wavelets. *Applied and Computational Harmonic Analysis*, 3(1):72 – 81.

[Kuo and Kaiser, 1966] Kuo, F. F. and Kaiser, J. F. (1966). *System analysis by digital computer*. Wiley.

[Nadeau and Johnson, 1998] Nadeau, R. M. and Johnson, L. R. (1998). Seismological studies at parkfield vi: Moment release rates and estimates of source parameters for small repeating earthquakes. *Bulletin of the Seismological Society of America*, 88(3):790–814.

[NVIDIA Corporation, 2018] NVIDIA Corporation (2018). NVIDIA CUDA C programming guide. Version 10.1.243.

[Oancea et al., 2014] Oancea, B., Andrei, T., and Dragoescu, R. M. (2014). GPGPU computing. *arXiv:1408.6923*.

[Oppenheim, 2000] Oppenheim, A. (2000). *Tratamiento de señales en tiempo discreto*. Fuera de colección Out of series. Pearson Educación.

[Oppenheim et al., 1998] Oppenheim, A., Willsky, A., Nawab, S., and Hernández, G. (1998). *Señales y sistemas*. Pearson Educación. Pearson Educación.

[Reinders, 2007] Reinders, J. (2007). *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. .ºReilly Media, Inc.".

[Rojas et al., 2014] Rojas, H. E., CortÃ, C. A., and RamÃrez, D. F. (2014). ImplementaciÃComputacional de la Transformada Fraccional de Fourier Discreta. *InformaciÃtecnolÃ*, 25:143 – 156.

[Ronen et al., 2001] Ronen, R., Mendelson, A., Lai, K., Lu, S.-L., Pollack, F., and Shen, J. P. (2001). Coming challenges in microarchitecture and architecture. *Proceedings of the IEEE*.

[Sanders and Kandrot, 2010] Sanders, J. and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition.

[Santhosh and Thomas, 2013] Santhosh, L. and Thomas, A. (2013). Implementation of radix 2 and radix 22 fft algorithms. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, volume 4, pages 1–4.

[Schmidt, 2013] Schmidt, A. L. (2013). Fft: Transformada rápida de fourier. pages 1–3.

[Storti and Yurtoglu, 2016] Storti, D. and Yurtoglu, M. (2016). *CUDA for engineers*. Addison-Wesley Professional, 1st edition.

[Uchida, 2019] Uchida, N. (2019). Detection of repeating earthquakes and their application in characterizing slow fault slip. *Progress in Earth and Planetary Science*, 6(1):40.

[Uchida and Bürgmann, 2019] Uchida, N. and Bürgmann, R. (2019). Repeating earthquakes. *Annual Review of Earth and Planetary Sciences*, 47(1):306–311, 318–323.

[Vidale et al., 1994] Vidale, J., EllIsworth, W., Cole, A., and Marone, C. (1994). Variations in rupture process with recurrence interval in a repeated small earthquake. *Nature*, 368:624–26.

[von Neumann, 1993] von Neumann, J. (1993). First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*

[Wilson et al., 2012] Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K., Mitchell, I. M., Plumbley, M., Waugh, B., White, E. P., and Wilson, P. (2012). Best practices for scientific computing.

[Zechar and Nadeau, 2012] Zechar, J. D. and Nadeau, R. M. (2012). Predictability of repeating earthquakes near Parkfield, California. *Geophysical Journal International*, 190(1).

[Ziavras, 2012] Ziavras, S. G. (2012). History of computation.