ESTRATEGIA COMPUTACIONAL PARA REDUCIR TIEMPOS DE EJECUCIÓN EN LA PROPAGACIÓN DE UNA ONDA DE ELECTROMAGNÉTICA 3D UTILIZANDO GPU'S Y FDTD

WILLIAM ESTEBAN LADINO CÁCERES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍA FÍSICO-MECÁNICA
ESCUELA DE INGENIERÍA ELÉCTRICA,
ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

ESTRATEGIA COMPUTACIONAL PARA REDUCIR TIEMPOS DE EJECUCIÓN EN LA PROPAGACIÓN DE UNA ONDA DE ELECTROMAGNÉTICA 3D UTILIZANDO GPU'S Y FDTD

WILLIAM ESTEBAN LADINO CÁCERES

Tesis de grado para optar al título de Ingeniero Electrónico

Director
Dr.Ing. JHEYSTON OMAR SERRANO LUNA.
Codirectora
Ph.D.ANA BEATRIZ RAMÍREZ SILVA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍA FÍSICO-MECÁNICA
ESCUELA DE INGENIERÍA ELÉCTRICA,
ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

Agradecimientos

Quiero agradecer a mi madre, padre y todas las personas que me apoyaron y guiaron a lo largo de la carrera y el desarrollo de este proyecto. Un agradecimiento especial a los profesores Jheyston Serrano y Ana Ramírez quienes estuvieron supervisando y apoyando el trabajo realizado.

CONTENIDO

pág.

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: ESTRATEGIA COMPUTACIONAL PARA REDUCIR TIEMPOS DE EJECUCIÓN EN LA PROPAGACIÓN DE UNA ONDA DE ELECTROMAGNÉTICA 3D UTILIZANDO GPU'S Y FDTD [*]

AUTOR: WILLIAM ESTEBAN LADINO CÁCERES [**]

PALABRAS CLAVE: HPC, GPU, DESCOMPOSICIÓN DE DOMINIO, TRANSFERENCIAS ASINCRÓNICAS, STREAMS, PROGRAMACIÓN EN PARALELO, CUDA, ONDA ELECTROMAGNÉTICA.

DESCRIPCIÓN:

La simulación de modelos físicos, matemáticos y geológicos son de gran importancia porque permiten predecir el comportamiento de sistemas complejos. Entre los modelos simulados más utilizados destacan, la propagación de ondas electromagnéticas, fluidos, diferencias finitas, elementos finitos, optimización, geotérmicos, terremotos, entre otros. La simulación de una onda electromagnética requiere de grandes cantidades de memoria RAM y amplios tiempos de ejecución, por ello en esta investigación se busca optimizar el uso de estos recursos a partir de una estrategia computacional y de transferencias asincrónicas. En este sentido, primero, se realiza una discretización de la ecuación de onda electromagnética 3D utilizando el método FDTD; segundo, a partir de la discretización se genera un algoritmo para la propagación de la onda; tercero, se implementa la primera estrategia, que consiste en la descomposición de dominio programada en dos GPU's conectadas a través del puerto PCI Express por medio de MPI; cuarto, se aplica la segunda estrategia en la cual se utilizan transferencias asincrónicas. Los resultados muestran que la primera estrategia presenta mejoras en los tiempos de ejecución de hasta un 32 % y permite que la simulación se lleve a cabo sobre dominios del doble de la capacidad permitida por la memoria RAM, en comparación con la ejecución de una sola GPU. La segunda estrategia permite mejorar un 10 % el tiempo de ejecución con respecto a la anterior.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Dr.Ing.Jheyston Omar Serrano Luna y codirectora Ph.D.Ana Beatriz Ramírez Silva.

ABSTRACT

TITLE: COMPUTATIONAL STRATEGY TO REDUCE EXECUTION TIMES IN 3D ELECTROMAGNETIC WAVE PROPAGATION USING GPU'S AND FDTD [*]

AUTHOR: WILLIAM ESTEBAN LADINO CÁCERES [**]

KEYWORDS: HPC, GPU, DOMAIN DECOMPOSITION, ASYNCHRONOUS TRANSFERS, STREAMS, PARALLEL PROGRAMMING, CUDA, ELECTROMAGNETIC WAVE.

DESCRIPTION:

The simulation of physical, mathematical, and geological models is of great importance because it allows predicting the behavior of complex systems. Among the most commonly used simulated models are electromagnetic wave propagation, fluids, finite differences, finite elements, optimization, geothermal, earthquakes, among others. The simulation of an electromagnetic wave requires large amounts of RAM memory and lengthy execution times. Therefore, this research seeks to optimize the use of these resources through a computational strategy and asynchronous transfers. First, the 3D electromagnetic wave equation is discretized using the FDTD method; second, an algorithm for wave propagation is generated from the discretization; third, the first strategy is implemented, which consists of domain decomposition programmed on two GPUs connected through the PCI Express port via MPI; fourth, the second strategy is applied, which uses asynchronous transfers. The results show that the first strategy presents improvements in execution times of up to $32\,\%$, and allows the simulation to be carried out on domains twice the capacity allowed by the RAM memory, compared to the execution on a single GPU. The second strategy improves execution times by $10\,\%$ compared to the previous one.

---

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Dr.Ing.Jheyston Omar Serrano Luna y codirectora Ph.D.Ana Beatriz Ramírez Silva.

INTRODUCCIÓN

Las unidades de procesamiento gráfico (GPU's) son excelentes para procesar grandes cantidades de datos debido a su arquitectura paralela y alta velocidad de procesamiento. Lo que les permite analizar información mucho más rápido que una CPU convencional. Esto es especialmente útil en aplicaciones que requieren el tratamiento de grandes cantidades de información. Por ejemplo, en la simulación de ondas electromagnéticas se procesan grandes cantidades de datos debido a la complejidad y la variedad de factores que las afectan, como: la frecuencia, la amplitud, la dirección, la interacción con diferentes materiales y la geometría del entorno. Además, las ondas electromagnéticas pueden propagarse en diferentes medios y ser reflejadas, refractadas y difractadas por objetos en su camino, lo que aumenta la complejidad de la simulación. Un ejemplo de esto se puede encontrar en el artículo de Robin M. Weiss y Jeffrey Shragge [1], donde se muestra que una implementación paralela de la ecuación de onda elástica 2D y 3D programada en una GPU logra aceleraciones de 10 y 28 veces, respectivamente.

Las propagaciones de ondas electromagnéticas nos permiten tener una respuesta del comportamiento de la onda en el subsuelo, esto nos brinda información de los materiales internos. Existen métodos geofísicos como el caso de GPR (Radar de Penetración terrestre) que permiten realizar adquisiciones no invasivas para identificar objetos enterrados utilizando ondas electromagnéticas 3D. El método GPR utiliza una antena transmisora para emitir un pulso electromagnético en un medio y una antena receptora para captar el campo eléctrico que se produce por los diferentes fenómenos físicos [2].

---

[1] Robin M. Weiss y Jeffrey Shragge. "Solving 3D anisotropic elastic wave equations on parallel GPU devices". En: *GEOPHYSICS* 78.2 (2013), F7-F15. DOI: 10.1190/geo2012-0063.1. eprint: https://doi.org/10.1190/geo2012-0063.1.

[2] Gregory Baker, Thomas Jordan y Jennifer Pardy. "An introduction to ground penetrating radar (GPR)". En: vol. 432. Ene. de 2007, págs. 1-18. DOI: 10.1130/2007.2432(01).

Para la estimación de los parámetros electromagnéticos es necesario disponer de un software de propagación de ondas electromagnéticas que permitan simular el comportamiento del campo medido por el equipo.

En este trabajo se implementa una ecuación de onda electromagnética 3D utilizando diferencias finitas en el dominio del tiempo. La solución de una ecuación diferencial requiere de millones de operaciones aritméticas, esto tiene un impacto en los tiempos de ejecución y recursos lógicos [3]. Para reducir estos tiempos de ejecución se propone utilizar una arquitectura de alto desempeño como las GPU's y una estrategia computacional para las transferencias de información entre ellas. La descomposición de dominio consta de tres etapas: dividir el dominio en dos partes iguales, agregar una zona de transferencias entre los dominios cuyo ancho depende del orden de las diferencias finitas y transferir constantemente los datos de la GPU0 hacia la GPU1 y viceversa, a través del puerto PCI Express [4].

Existen diferentes maneras para programar en paralelo, para esta investigación se utilizó CUDA (Arquitectura Unificada de Dispositivos de Cómputo). Se compone de un compilador y un conjunto de herramientas que permiten usar CUDA C (variación del lenguaje de programación C) [4]. El flujo de procesamiento de CUDA, consiste en: mover los datos de entrada del host al device [5], ejecutar el kernel en el device y mover el resultado de device de regreso al host [4].

---

[3] J.E. Houle y D.M. Sullivan. *Electromagnetic Simulation Using the FDTD Method with Python*. Wiley, 2020.

[4] J. Cheng, M. Grossman y T. McKercher. *Professional CUDA C Programming*. EBL-Schweitzer. Wiley, 2014.

[5] Dentro de CUDA se conoce como host a la CPU, device a la GPU, kernel a una función de CUDA y nt a una muestra de tiempo discreto

Las principales contribuciones de este trabajo son tres. En primer lugar, se presenta la propagación de la onda usando dos GPU's por medio una interfaz de paso de mensajes (MPI) y una descomposición de dominio. En segundo lugar, la aplicación de kernels con streams y transferencias asincrónicas, y en tercer lugar presentamos un análisis de rendimiento de las implementaciones propuestas, en términos de tiempo de ejecución y memoria RAM. El esquema de este documento es el siguiente: en la sección uno se presenta la introducción; en la sección dos el marco teórico del modelo de propagación de la onda electromagnética; en la sección tres, se describe la implementación en una GPU: en la sección cuarta y quinta, se describen e implementan las estrategias para mejorar el rendimiento de la propagación; finalmente, en las secciones seis y siete, se presentan los resultados y conclusiones, respectivamente.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

- Desarrollar e implementar una estrategia computacional que permita reducir tiempos de ejecución en la simulación de propagaciones de ondas electromagnéticas 3D en un medio homogéneo, no dispersivo y con pérdidas.

1.2. OBJETIVOS ESPECÍFICOS

- Implementar una ecuación de onda electromagnética 3D utilizando diferencias finitas sobre una GPU.
- Desarrollar la estrategia computacional que permita reducir los tiempos de ejecución
- Medir la mejora en tiempos de la implementación de la estrategia computacional.

2. MODELO DE PROPAGACIÓN DE LA ONDA

En esta sección se describe la ecuación para una onda que se propaga en un medio 3D isotrópico, su discretización por el método FDTD, las ecuaciones de la fuente y el modelo de permitividad relativa.

2.1. Ecuación onda

La ecuación de onda electromagnética [6] para el campo eléctrico es,

$$\nabla^2 E - \mu\epsilon\frac{\partial^2 E}{\partial t^2} = s_{(Cx,Cy,Cz)}, \tag{1}$$

donde ∇ (nabla) representa un operador diferencial vectorial; E representa la intensidad de campo Eléctrico; μ la permeabilidad magnética; ϵ la permitividad eléctrica; s la fuente; Cx, Cy y Cz representan las coordenadas espaciales para los ejes x, y y z, respectivamente. La ecuación de onda puede volverse más complicada en medios complejos, dispersivos, anisótropos y no lineales, pero, el enfoque de esta investigación es la implementación de la estrategia computacional.

2.2. FDTD

El método **FDTD** se ha seleccionado para discretizar la ecuación de onda. Este es un método sencillo de aplicar para problemas electromagnéticos, en comparación con otros métodos como: el método de los momentos, los métodos de volumen finito, los métodos de elementos finitos y los métodos espectrales [6]. Pese a su fácil aplicación este método requiere una gran cantidad de memoria RAM y tiempos de ejecución [6].

---

[6] U.S. Inan y R.A. Marshall. *Numerical Electromagnetics: The FDTD Method*. Cambridge University Press, 2011.

En la ecuación (2), se observa la definición de una diferencia finita centrada con el método FDTD [6]. En esta se presenta punto futuro f_i^{n+1}, un punto presente f_i^n y uno pasado f_i^{n-1}, todos en dirección *n*, además de Δt que representa la discretización del tiempo, de la misma manera en las derivadas espaciales se utilizó un Δ en las tres dimensiones x, y, z de 0.2,

$$\frac{\partial f^2}{\partial t^2}\Big|_i^n \simeq \frac{f_i^{n+1} - 2f_i^n + f_i^{n-1}}{\Delta t^2}, \tag{2}$$

siguiendo la metodología aplicada al ejemplo anterior, aplicamos el método de diferencias finitas a la ecuación (1),

$$\nabla^2 E = \mu\epsilon \left[\frac{E^{n+1} - 2E^n + E^{n-1}}{\Delta t^2}\right] + s_{(Cx,Cy,Cz)}, \tag{3}$$

finalmente despejamos E^{n+1} de la ecuación (3) con la que se modelará la propagación de la onda, ecuación (4).

$$E^{n+1} = \frac{\nabla^2 E \cdot \Delta t^2}{\mu\epsilon} + 2E^n - E^{n-1} + s_{(Cx,Cy,Cz)}, \tag{4}$$

Por consiguiente, para desarrollar el algoritmo de propagación de la onda electromagnética, se debe resolver los cuatro términos de la ecuación (4).

1. $\frac{\nabla^2 E \cdot \Delta t^2}{\mu\epsilon} \leftarrow \frac{\Delta t^2}{\mu\epsilon}\left[\frac{\partial^2 E}{\partial x^2} + \frac{\partial^2 E}{\partial y^2} + \frac{\partial^2 E}{\partial z^2}\right]$,
2. $2E^n$,
3. E^{n-1},
4. $s_{(Cx,Cy,Cz)}$.

Primero, se aplica el operador nabla cuadrado a la intensidad de campo eléctrico ($\nabla^2 E$), por consiguiente, se producen las derivadas parciales espaciales de segundo orden en las tres coordenadas. Para resolver cada derivada parcial se aplica el método de diferencias finitas

espaciales, como son derivadas espaciales se aplica una derivada hacia atrás (Backward finite difference) ecuación (5) y una derivada parcial hacia adelante (Forward finite difference) ecuación (6), donde din representa el campo eléctrico de entrada E y tid representa el índice de los kernels. Los coeficientes de las diferencias finitas espaciales son $\frac{9}{8}$ y $\frac{1}{24}$ se obtuvieron del artículo de Yang Liu y Mrinal K. Sen [7]; segundo, este término representa el tiempo presente de la intensidad de campo eléctrico de modo que no se le aplica ningún cambio; tercero, E^{n-1} hace referencia a un instante de tiempo pasado discreto, este se obtiene respaldando E antes de que cambie el ciclo de tiempo;

$$\frac{\partial E}{\partial x}\Big|^{tid} \simeq \frac{\frac{9}{8}\left(din^{tid} - din^{tid-1}\right) - \frac{1}{24}\left(din^{tid+1} - din^{tid-2}\right)}{\Delta x}, \tag{5}$$

$$\frac{\partial E}{\partial x}\Big|^{tid} \simeq \frac{\frac{9}{8}\left(din^{tid+1} - din^{tid}\right) - \frac{1}{24}\left(din^{tid+2} - din^{tid-1}\right)}{\Delta x}, \tag{6}$$

cuarto, consiste en generar la fuente a través de las ecuaciones (7), utilizadas para generar la forma del estímulo, donde f es la frecuencia (f = 100[Mhz]). La Figura 2.1 representa la firma de la ondícula (ricker) en dominio del tiempo, esta es utilizada en todas las simulaciones. Se eligió este tipo de pulso y frecuencia debido a su uso común en aplicaciones de GPR [8].

$$\begin{aligned} &a = (\pi f)^2, t_0 = \left\lceil \frac{\Delta t}{f} + 20 \right\rceil \Delta t, \\ &s[nt] = \left(1 - 2a\left(nt\Delta t - t_0\right)^2\right) e^{-a(nt\Delta t - t_0)^2}. \end{aligned} \tag{7}$$

---

[7] Yang Liu y Mrinal K. Sen. "An implicit staggered-grid finite-difference method for seismic modelling". En: *Geophysical Journal International* 179.1 (oct. de 2009). DOI: 10.1111/j.1365-246X.2009.04305.x. eprint: https://academic.oup.com/gji/article-pdf/179/1/459/5906542/179-1-459.pdf.

[8] Dorin Grigoras www.stepofweb.com. *Georadar de Penetración terrestre*.

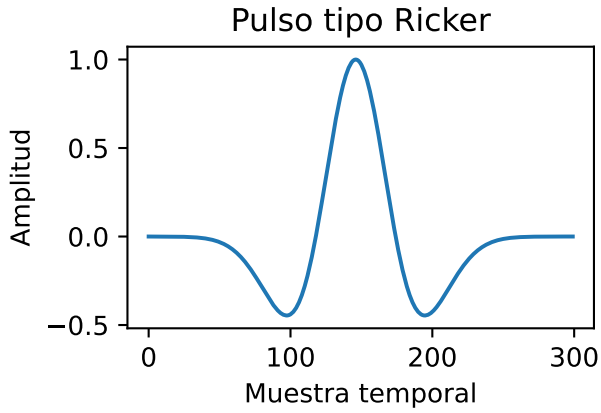

Figura 2.1. Fuente implementada en todas las propagaciones tipo Ricker

2.3. Modelo de permitividad relativa

Para la propagación de la onda se plantearon dos modelos: el primer modelo se presenta en la Figura 2.2, donde se tiene un objeto enterrado y se aprecia que la onda electromagnética choca con el objeto y parte de su energía se refleja y retorna a la superficie. El segundo modelo no contiene el objeto enterrado y el valor de $\epsilon_r = 1$ [9].

Por otra parte, las ecuaciones necesarias para la implementación de las CPML fueron tomadas del artículo de Damir Pasalic y Ray McGarry [10]. Para el algoritmo se definieron con un ancho de 20 puntos en cada dimensión, para evitar cualquier tipo de reflexión.

---

[9] ϵ_r es una constante eléctrica llamada permitividad relativa que describe la capacidad de un material a ser polarizado por un campo eléctrico

[10] Damir Pasalic y Ray McGarry. "Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations". En: *SEG Technical Program Expanded Abstracts 2010*. DOI: 10.1190/1.3513453. eprint: https://library.seg.org/doi/pdf/10.1190/1.3513453.

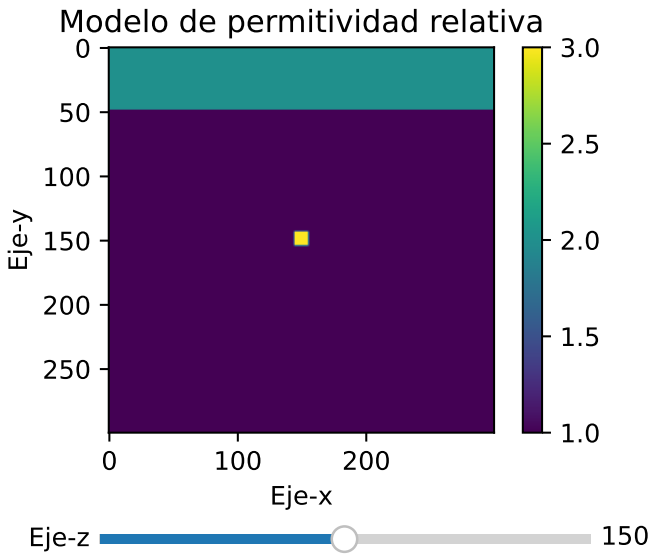

Figura 2.2. Corte en el eje-z del modelo de permitividad relativa sobre el cual se realizó la propagación

3. IMPLEMENTACIÓN EN UNA GPU

A continuación, se presenta el algoritmo 1 que representa la propagación de la onda electromagnética.

Algoritmo 1: Una GPU

1: $EC3D(cub, s, \Delta t, \Delta(x, y, z), \mu, \epsilon)$ ▷ EC3D Entradas
2: s ▷ Fuente
3: $h_{in} \leftarrow cub.bin$ ▷ Modelo de entrada
4: $E \leftarrow 0$ ▷ Intensidad del campo presente
5: $E^{n+1} \leftarrow 0$ ▷ Intensidad del campo futuro
6: $E^{n-1} \leftarrow 0$ ▷ Intensidad del campo pasado
7: $\epsilon_r \leftarrow h_{in_{dev}}$ ▷ Trasferencia del modelo a device
_ **for** $n \leftarrow 0$, n_t **do**
n,número de iteraciones de EC3D
8: $s_{dev} \leftarrow s$ ▷ Fuente, host a device
9: $\nabla^2 E_{dev} \leftarrow \frac{\partial^2 E}{\partial x^2} + \frac{\partial^2 E}{\partial y^2} + \frac{\partial^2 E}{\partial z^2}$ ▷ Laplaciano
10: $E^{n+1}_{dev} \leftarrow \frac{\nabla^2 E \cdot \Delta t^2}{\mu\epsilon} + 2E^n - E^{n-1}$
11: $E^{n+1}_{dev} \leftarrow E^{n+1} + s_{(Cx,Cy,Cz)}$ ▷ Se incluye la fuente s
12:
13: $E^{n+1}_{dev} \leftarrow E^{n+1}_{host}$
14: **return** E^{n+1}_{host}

La propagación de la onda fue realizada utilizando una tarjeta gráfica tesla k40c que cuenta con las características presentadas en la Tabla 3.1.

3.1. Onda propagada

En esta subsección se presentan dos instantes de tiempo discreto de la onda propagada sin barreras absorbentes: Figura 3.1 y Figura 3.2. Mientras que la Figura 3.3 presenta la propagación de la onda con barreras absorbentes.

Tabla 3.1. Características tesla k40c

Item	Tesla k40c
Architecture:	Kepler
Bus interface:	PCI-E 3.0 x 16
Base Clock	745 MHz
Boost Clock	876 MHz
Memory Clock	1502 MHz
Shading Units	2880
TMUs	240
ROPs	48
SMX Count	15
L1 Cache	16 KB (per SMX)
L2 Cache	1536 KB
Memory Size	12 GB
Memory Type	GDDR5
Memory Bus	384bit
Bandwidth	288,4 GB/s

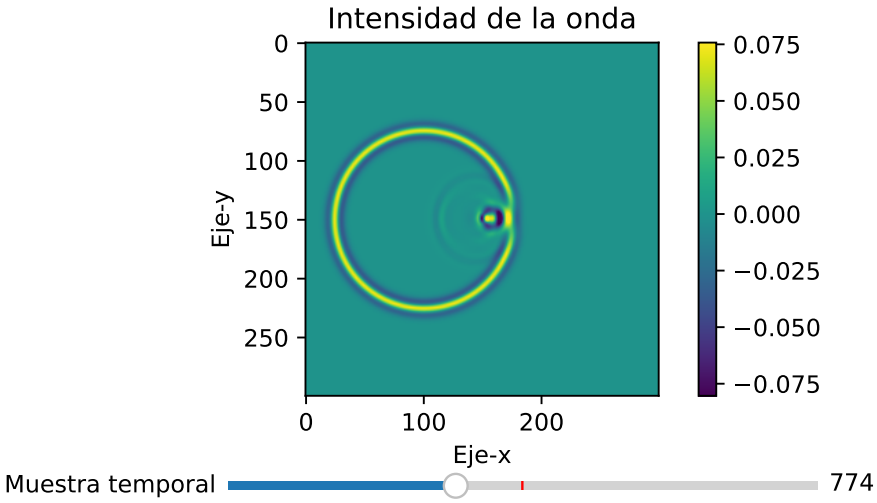

Figura 3.1. Propagación de la onda electromagnética sobre el modelo de permitividad relativa presentado en la Figura 2.2, sin barreras absorbentes y con la fuente en las coordenadas, *Cx = 100, Cy = 150 y Cz= 150*

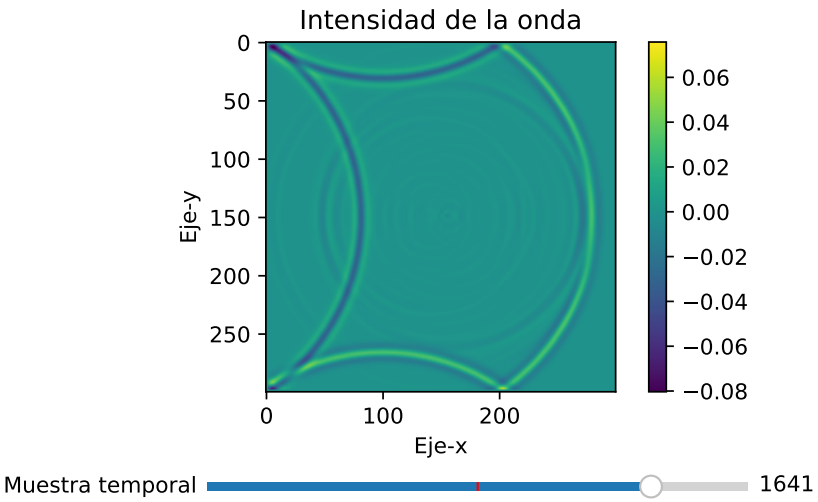

Figura 3.2. Propagación de la onda electromagnética sobre el modelo de permitividad relativa presentado en la Figura 2.2, sin barreras absorbentes y y con la fuente en las coordenadas, *Cx = 100, Cy = 150 y Cz= 150*

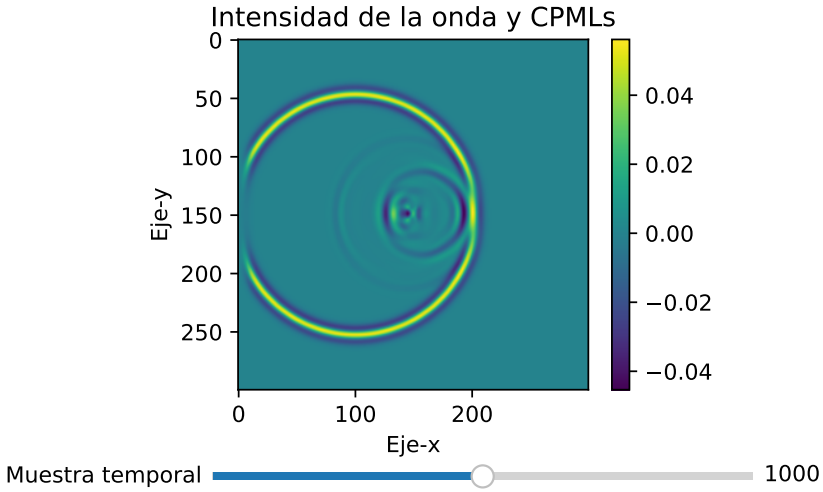

Figura 3.3. Propagación de la onda electromagnética sobre el modelo de permitividad relativa presentado en la Figura 2.2 con barreras absorbentes y con la fuente en las coordenadas, *Cx = 100, Cy = 150 y Cz= 150*

4. DESCOMPOSICIÓN DE DOMINIO

Cuando se trabaja con un dominio grande, la información que conduce a la solución requiere tiempos largos y grandes cantidades de memoria. La descomposición de dominio consiste en dividir el área de trabajo en diferentes partes. La Figura 4.1 presenta el trabajo que puede ser realizado mediante varias unidades de cómputo, como computadores secuenciales o arquitecturas paralelas, sin embargo, existe una limitación que depende del dispositivo con el que se esté trabajando "GPU's". Cuando se trabaja con varias GPU's se puede genera un cuello de botella producido por la comunicación entre ellas y el host, ya que estas no están conectadas directamente, por lo que los datos deben pasar de la GPU1 al host y del host a la GPU2.

Para el desarrollo de esta estrategia computacional se utilizaron dos tarjetas gráficas **tesla k40c** conectadas por el puerto PCI Express, que tiene un ancho de banda de 16 GB/s, sus características se presentan en la Tabla 3.1. Para descomponer el dominio se determinó partir el área de trabajo en dos partes de igual tamaño, *DOMINIO A y DOMINIO B* como se observa en la Figura 4.1.

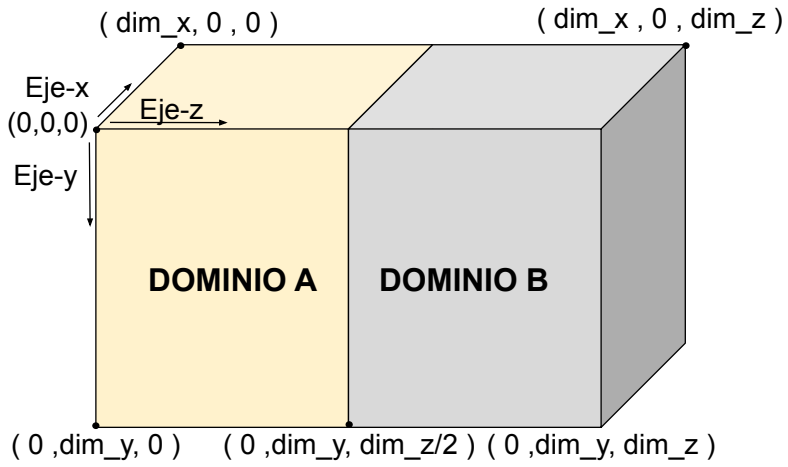

Figura 4.1. Descomposición del dominio en dos partes con corte perpendicular sobre Eje-z

Si no se considera una comunicación bidireccional entre los dominios, la propagación quedará atrapada en una sola parte, como se observa en la Figura 4.2.

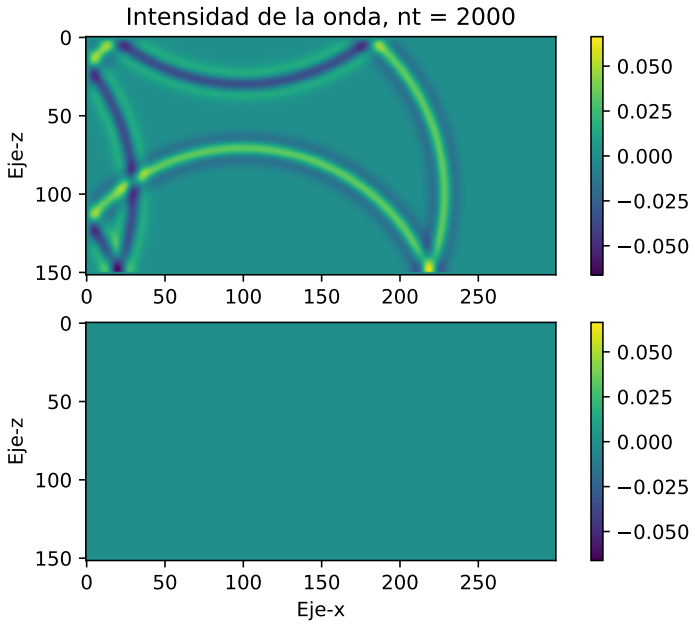

Figura 4.2. Propagación de la onda electromagnética en dos dominios, sin capas de comunicación, sobre el modelo de permitividad relativa ideal = 1, sin barreras absorbentes y con la fuente ubicada en las coordenadas *Cx = 100, Cy = 150 y Cz = 100*

Las operaciones como las derivadas espaciales, que describen parte matemática de la onda necesitan valores posteriores o anteriores dependiendo del orden de la derivada como se observa en las ecuaciones (5) y (6), por lo que el espacio de propagación se reduce debido a esto cambia el dominio. La idea de la partición de dominio es obtener el mismo resultado que si se operara usando una sola unidad de cómputo, pero de una forma acelerada.

4.1. Implementación de descomposición de dominio en código

Para resolver el problema de la reducción del dominio por las derivadas de diferentes tipos y la comunicación entre ellos luego de la partición. Se realiza una extensión de los subdominios

que permite mantener el dominio original y sobre esta se realiza la comunicación bidireccional que los mantiene conectados. La extensión de dominio se puede observar en la Figura 4.3, esta consiste en anexar al final del *DOMINIO A* los primeros datos del *DOMINIO B* (flecha azul) y al inicio del *DOMINIO B* se anexan los últimos datos del *DOMINIO A* (flecha negra). El número de las capas anexadas *d*, al dominio depende del orden de las derivadas espaciales, es decir si se requieren dos puntos al futuro, se anexan dos columnas en dirección *z*.

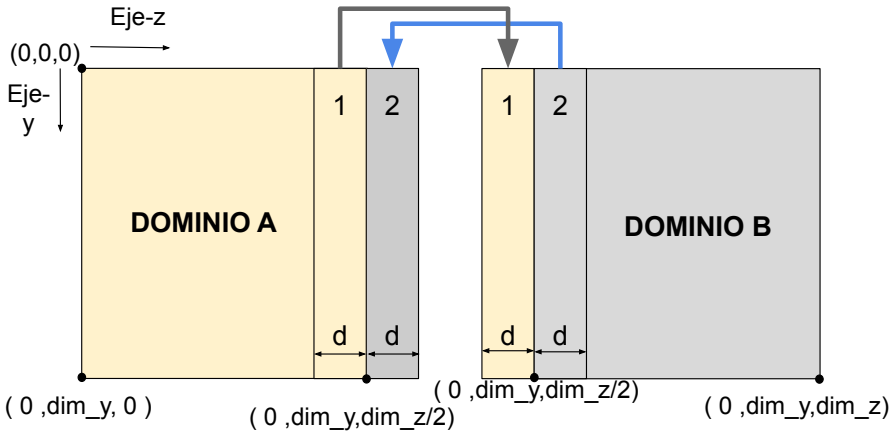

Figura 4.3. Implementación de las capas de comunicación en una vista 2D sobre el dominio dividido

El código 4.1 presenta la implementación propuesta para realizar las transferencias de un núcleo de la CPU a otro núcleo de la CPU. Este código se divide en dos pasos, en el primer paso desde el *DOMINIO A* se trasfieren los datos necesarios hacia el *DOMINIO B* y en el segundo paso trasferimos desde el *DOMINIO B* los datos necesarios al *DOMINIO A*. Estas trasferencias fueron hechas utilizando MPI, que es una especificación para programación de paso de mensajes, visto de otra forma, es una librería que proporciona funciones para comunicar datos entre diferentes procesos. MPI puede ser usada en lenguajes como C, C++ o Fortran [4]. Para el desarrollo de esta estrategia se utilizó el lenguaje de programación CUDA C ya que este permite hacer un mejor control del paralelismo de CUDA.

Código 4.1. Escrito en C para la comunicación de dos procesos por MPI en host

```
1  if (rank == 0){
2      MPI_Send(DOMINIO_A+dim_x*dim_y*(dim_z/2 -2),2*dim_x*dim_y,
           MPI_FLOAT,1,0, MPI_COMM_WORLD);}
3    else if (rank == 1){
4  MPI_Recv(DOMINIO_B,2*dim_x*dim_y,MPI_FLOAT,0,0,MPI_COMM_WORLD,
       MPI_STATUS_IGNORE);}
5
6    if (rank == 1){
7      MPI_Send(DOMINIO_B+dim_x*dim_y*2,2*dim_x*dim_y,MPI_FLOAT,0,1,
           MPI_COMM_WORLD);}
8    else if (rank == 0){
9      MPI_Recv(DOMINIO_A+dim_x*dim_y*(dim_z/2),2*dim_x*dim_y,
           MPI_FLOAT,1,1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);}
```

Cuando se desea transferir datos de una GPU a otra, es necesario pasar por el puerto PCI Express ya que estas no tienen una memoria compartida, por lo que primero se transfieren los datos al host como se observa en el código 4.2 y luego se trasfieren los datos entre los dominios como ocurre en el código 4.1. sub_capa es una variable temporal del host donde se almacena la transferencia.

Código 4.2. Escrito en C para la comunicación device a host en las dos GPU's

```
1
2  if (rank == 0){
3      launch_cudaMemcpy_DeviceToHost(sub_capa, E + (((dim_z/2)-2) *
           dim_x * dim_y), dim_x * dim_y * 2);
4  }
5
6  else if (rank == 1){
```

```
7       launch_cudaMemcpy_DeviceToHost(sub_capa, E + (2 * dim_x * dim_y
            ), dim_x * dim_y * 2);
8   }
```

4.2. Implementación en dos GPU's

Como ya se mencionó es necesario transferir **d** número de capas, donde **d** depende del orden de las derivadas. Para esta aplicación se transfieren dos capas en tres instantes diferentes del código como se observa en el algoritmo 2 en las líneas 8, 10, y 11. En la primera trasferencia se comunica el modelo de permitividad $\epsilon_{rGPU1} \rightleftharpoons \epsilon_{rGPU2}$, en la segunda se realiza la comunicación del campo E, $E_{GPU1} \rightleftharpoons E_{GPU2}$ y en la tercera se realiza la comunicación de las segundas derivadas en dirección z, $\frac{\partial^2 E}{\partial z^2}{}_{GPU1} \rightleftharpoons \frac{\partial^2 E}{\partial z^2}{}_{GPU2}$.

Algoritmo 2: Una GPU

	$EC3D(cub, s, \Delta t, \Delta(x, y, z), \mu, \epsilon)$	▷ EC3D Entradas
2:	s	▷ Fuente
	$h_{in} \leftarrow cub.bin$	▷ Modelo de entrada
4:	$E \leftarrow 0$	▷ Intensidad del campo presente
	$E^{n+1} \leftarrow 0$	▷ Intensidad del campo futuro
6:	$E^{n-1} \leftarrow 0$	▷ Intensidad del campo pasado
	$\epsilon_r \leftarrow h_{in_{dev}}$	▷ Trasferencia del modelo a device
8:	$\epsilon_{rGPU1} \rightleftharpoons \epsilon_{rGPU2}$	▷ Trasferencia MPI
_	**for** $n \leftarrow 0$, n_t **do** n,número de iteraciones de EC3D	
	$E_{GPU1} \rightleftharpoons E_{GPU2}$	▷ Trasferencia MPI
10:	$\frac{\partial^2 E}{\partial z^2}{}_{GPU1} \rightleftharpoons \frac{\partial^2 E}{\partial z^2}{}_{GPU2}$	▷ Trasferencia MPI
	$s_{dev} \leftarrow s$	▷ Fuente, host a device
12:	$\nabla^2 E_{dev} \leftarrow \frac{\partial^2 E}{\partial x^2} + \frac{\partial^2 E}{\partial y^2} + \frac{\partial^2 E}{\partial z^2}$	▷ Laplaciano
	$E_{dev}^{n+1} \leftarrow \frac{\nabla^2 E \cdot \Delta t^2}{\mu\epsilon} + 2E^n - E^{n-1}$	
14:	$E_{dev}^{n+1} \leftarrow E^{n+1} + s_{(Cx,Cy,Cz)}$	▷ Se incluye la fuente s
16:	$E_{dev}^{n+1} \leftarrow E_{host}^{n+1}$	
	return E_{host}^{n+1}	

Posteriormente se presenta la salida del algoritmo 2, Figura 4.4.

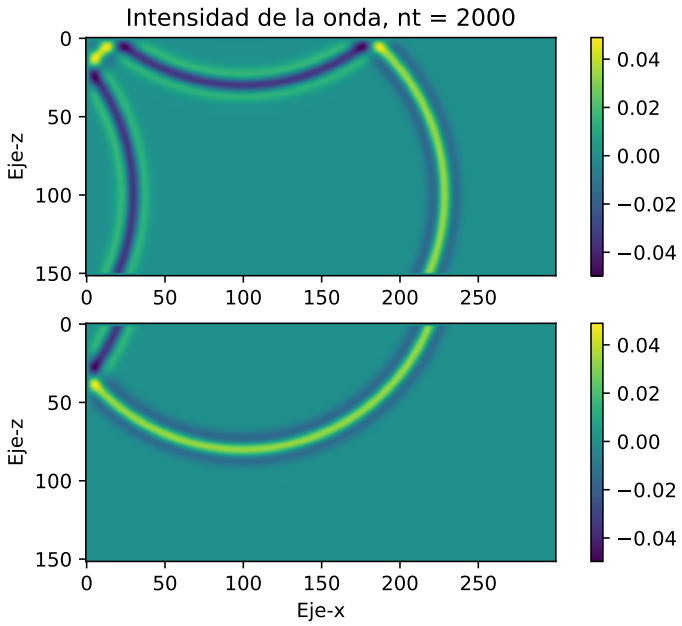

Figura 4.4. Propagación de la onda electromagnética en dos dominios, CON capas de comunicación, sobre el modelo de permitividad relativa ideal = 1, sin barreras absorbentes y con la fuente ubicada en las coordenadas *Cx = 100, Cy = 150 y Cz = 100*

5. CONCURRENCIA

Existen dos tipos de concurrencia en la programación de CUDA: un flujo serial y varios flujos en paralelo. En esta subsección se explorarán dos estrategias complementarias, el uso de los streams y los streams con transferencias asincrónicas. Debido a que todas las operaciones en cola en un flujo CUDA son asíncronas, es posible superponer su ejecución con otras operaciones en el sistema del dispositivo host. Esto permite ocultar el costo de realizar esas operaciones al realizar otro trabajo útil al mismo tiempo [4].

5.1. Streams

De acuerdo con el Guía de programación CUDA [11], un stream es una secuencia de comandos (posiblemente emitidos por diferentes subprocesos de host) que se ejecutan en orden. En la Figura 5.1 se observa el modo serial, en el cual todas las funciones y kernels se ejecutan una detrás de otra y en el modo concurrente se observa cómo se distribuyen los procesos de forma paralela en tres streams. Para poder ejecutar los streams de forma adecuada es necesario que los recursos estén disponibles y que el proceso del stream siguiente no utilice los recursos del stream anterior [4]. En el código 5.1 se presentan las declaraciones necesarias para la utilización de los streams [4].

Código 5.1. Escrito en CUDA C para las definiciones esenciales de los streams

```
1 int n_streams = 3;
2 cudaStream_t *streams;
3 streams =(cudaStream_t*)malloc(n_streams *sizeof(cudaStream_t));
4 for (int i=0;i < n_streams; i++) {cudaStreamCreate(&streams[i]);}
```

[11] Lei Mao. *Cuda Stream*. 2022.

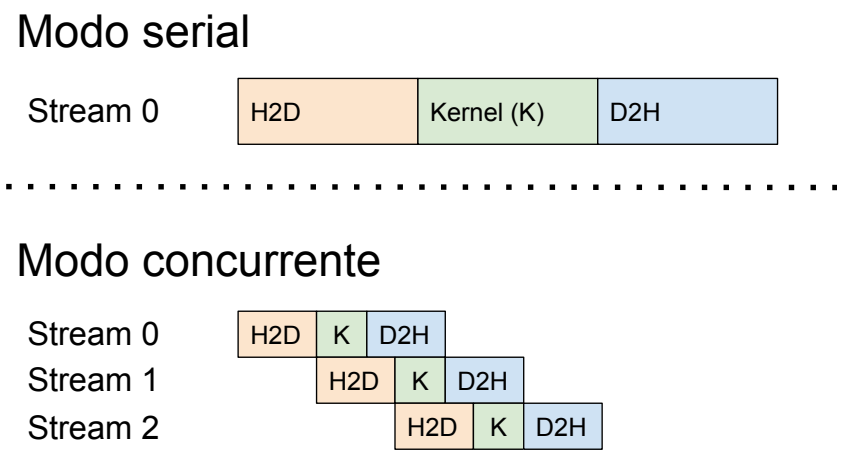

Figura 5.1. Modo serial vs modo concurrente, para dos transferencias y un kernel

5.1.1. Aplicación en una GPU Analizando el núcleo del flujo [12] de la propagación de la onda sin CPML, Figura 5.2,

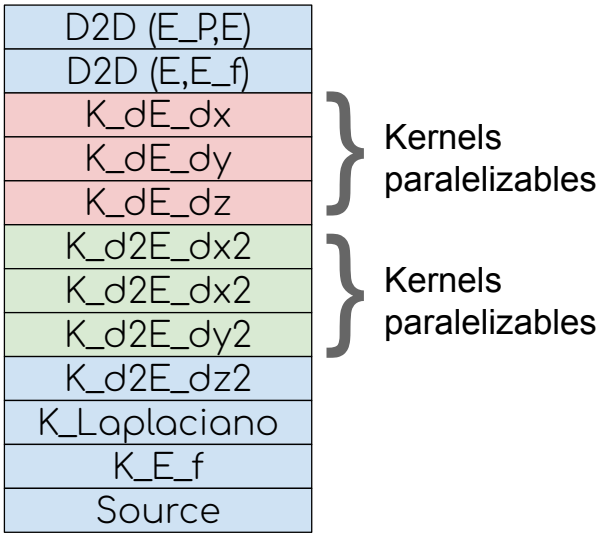

Figura 5.2. Selección de los kernels a los que se puede aplicar stream del flujo principal de la propagación de la onda en una GPU

[12] D2D = transferencia device to device, K = kernel, K_E_f= kernel donde se agrega la fuente como en la linea 12 del algoritmo 1

se observa que los kernels de las derivadas de primero y segundo orden no dependen entre sí, por lo que podemos ejecutarlos utilizando los streams como en la Figura 5.3.

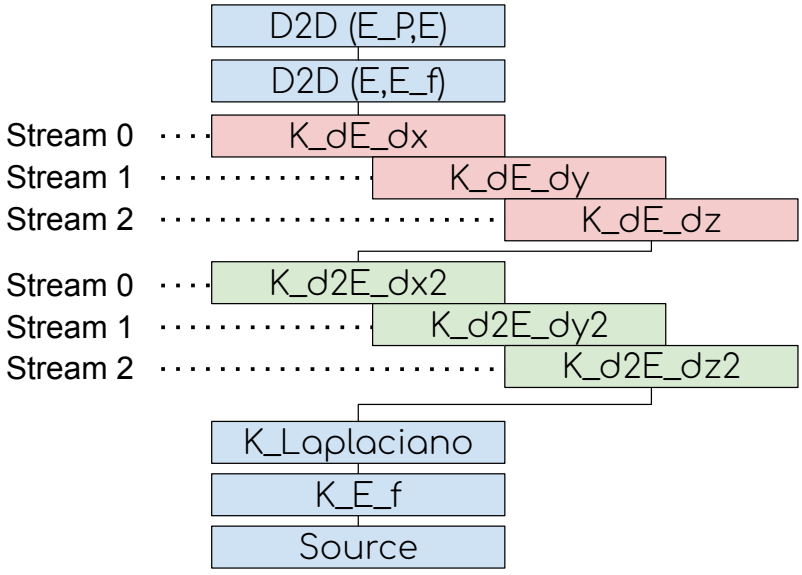

Figura 5.3. Flujo del código de la propagación en una GPU con los streams aplicados

5.1.2. Aplicación kernel Las dos propagaciones principales de esta investigación (propagación en una GPU y dos GPU's) se ejecutan distribuyendo los recursos (hilos y bloques), por cada elemento del dominio como en el código 5.2.

Código 5.2. Kernel tradicional escrito en CUDA C

```
1  __global__ void kernel_dE_dx(float32_t *d_in,float32_t *d_dE_dx,int
       dim_x,int dim_y,int dim_z,float d_x){
2  int ix=threadIdx.x+blockDim.x*blockIdx.x;
3  int iy=threadIdx.y+blockDim.y*blockIdx.y;
4  int iz=threadIdx.z+blockDim.z*blockIdx.z;
5  int tid =ix+dim_x*iy+dim_x*dim_y*iz;
6
7  if (ix > 1 && ix < (dim_x - 2) && iy > 1 && iy < (dim_y - 2) && iz
       > 1 && iz < (dim_z - 2)){
```

```
8
9  d_dE_dx[tid]=(1.0/d_x)*(weights_fdtd[0]*(d_in[tid]-d_in[tid-1])-
       weights_fdtd[1]*(d_in[tid+1]-d_in[tid-2]));
10   }}
```

Para ejecutar los **streams** es necesario cambiar la forma en la que se distribuyen los recursos, ya que de la forma definida en el código 5.2 se reservan todos los recursos incluso aquellos que no se van a utilizar. Para solucionar este problema se planteó otra estrategia para la ejecución de los kernels conocida como grid-stride loop. Esta estrategia consiste en definir los recursos en hilos y bloques y construir el kernel con un ciclo for que repite los recursos definidos hasta completar el dominio, esta implementación se observa en el código 5.3.

Código 5.3. Kernel tradicional convertido a un kernel cíclico y escrito en CUDA C

```
1  __global__ void kernel_dE_dx(float32_t *d_in, float32_t *d_dE_dx,
       int dim_x, int dim_y, int dim_z, float d_x){
2
3    int index_x=blockIdx.x*blockDim.x+threadIdx.x;
4    int stride_x=blockDim.x*gridDim.x;
5    int index_y=blockIdx.y*blockDim.y+threadIdx.y;
6    int stride_y=blockDim.y*gridDim.y;
7    int index_z =blockIdx.z*blockDim.z+threadIdx.z;
8    int stride_z =blockDim.z*gridDim.z;
9
10   for (int ix=index_x;ix<dim_x;ix+=stride_x){
11     for (int iy=index_y;iy<dim_y;iy+=stride_y){
12       for (int iz=index_z;iz<dim_z;iz+=stride_z){
13         int tid =ix+dim_x*iy+dim_x*dim_y*iz;
14
15 if(ix>1&&ix<(dim_x-2)&&iy>1&&iy<(dim_y-2)&&iz>1&&iz<(dim_z-2)){
```

```
16
17  d_dE_dx[tid]=(1.0/d_x)*(weights_fdtd[0]*(d_in[tid]-d_in[tid-1])-
        weights_fdtd[1]*(d_in[tid+1]-d_in[tid-2]));
18  }}}}}
```

5.2. Trasferencias asincrónicas

Antes de ejecutar el resultado de una operación asincrónica se debe verificar que la operación se haya completado usando las APIs de CUDA [4]. Si bien las operaciones dentro del mismo flujo CUDA tienen un orden estricto, las operaciones en diferentes flujos no tienen restricción en el orden de ejecución. Mediante el uso de varios flujos para lanzar núcleos simultáneos se puede implementar simultaneidad a nivel de cuadrícula [4].

Para realizar una transferencia de forma asíncrona es necesario: especificar el nombre la función $cudaMemcpyAsync$, un parámetro extra que indique el proceso que realice la transferencia $cudaStream_t$, y definir memoria de host anclada. A continuación, se presentan las lineas de código necesarias, código 5.4.

Código 5.4. Definiciones de memoria anclada para el uso de las transferencias asincrónicas escitas en CUDA C

```
1  cudaMallocHost(void *ptr, size_t size);
2
3  cudaHostAlloc(void *pHost, size_t size, unsigned int_flags);
4
5  cudaMemcpyAsync(void* dst, void* src, size_t count, cudaMemcpyKind
       kind, cudaStream_t stream = 0);
```

Considerando las estrategias anteriores solo es útil implementar las transferencias asincrónicas en la última transferencia de MPI, ya que el proceso siguiente K_d2E_dx2 no depende de esta transferencia, los kernels y la transferencia apta se observan en la Figura 5.4 en los recuadros rojos y las transferencias de MPI en los recuadros verdes.

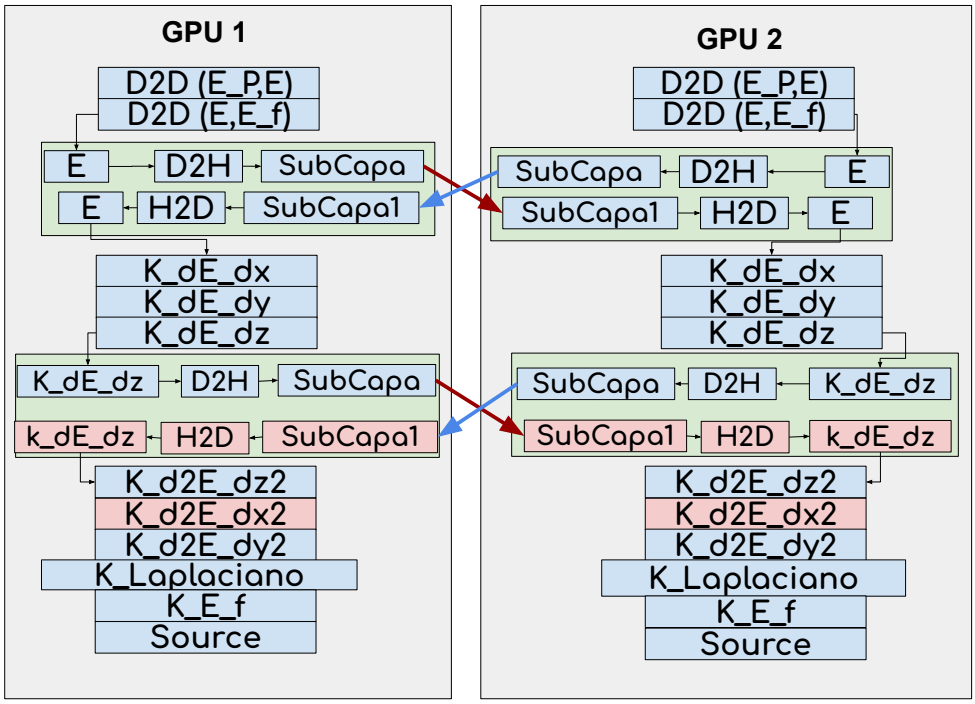

Figura 5.4. Flujo del código de la propagación dos GPU's con los streams aplicados y selección de transferencias que se pueden volver asíncronas

6. RESULTADOS

Para la medición del rendimiento de la estrategia computacional se hicieron dos pruebas en la cuales se varió la dimensión del dominio en dos formas diferentes. Los datos del tiempo de ejecución se toman utilizando nvprof y procesados mediante Python, para las mediciones de los streams se utiliza nvvp de CUDA ya que gráficamente permite visualizar los kernels, los tiempos y su paralelismo y para la medición del rendimiento de las transferencias asincrónicas se utiliza un bucle que ejecuta el algoritmo 100 veces y registra su tiempo en un archivo con el que se obtuvo un promedio de tiempo en segundos.

6.1. Prueba uno, comparación de tiempos y dominio límite

El objetivo de esta prueba es comparar los tiempos de ejecución y los límites del dominio, para la propagación en una y dos GPU's. En esta prueba se varió el dominio de forma constante, es decir, la misma variación en todas las dimensiones, las dimensiones usadas se observan en la primera columna de la Tabla 6.1.

Tabla 6.1. Resultados 1, comparativa de tiempos de ejecución entre la propagación en una y dos GPU's con variaciones en los tres Ejes

Dim_x,y,z iguales	**Tiempo[s] usando dos GPU's**	**Tiempo[s] usando una GPU**	**comparación de tiempos [s]**	**% de Mejora**
50	0,104	0,115	0,011	9,694
100	0,641	0,859	0,218	25,362
200	4,377	6,239	1,862	29,844
300	14,891	21,769	6,878	31,594
400	33,58	48,456	14,876	30,700
500	67,497	99,987	32,49	32,494
600	120,784			

Como se observa en la última fila del recuadro hay casillas vacías, esto se debe a que el total de la memoria RAM de una GPU de 12 GB se ocupó por completo, lo que denota una ventaja en la implementación de la estrategia, la cual permite trabajar con un dominio del doble al que se trabaja con una sola GPU. En dos GPU's dimz*dimx*dimy = (600)(600)(600) = 216.000.000 y en una GPU se alcanzó (500)(500)(500) = 125.000.000. las mejoras en tiempo son de hasta un 32.5 % y varían dependiendo del tamaño del dominio, que como se observa a mayor sea el dominio, mejor el rendimiento. Los resultados de la Tabla 6.1 se pueden visualizar en la Figura 6.1.

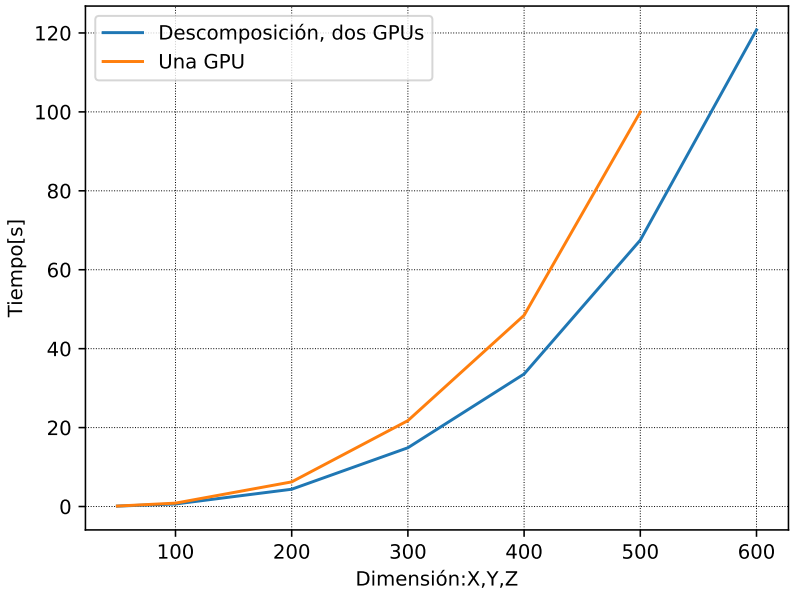

Figura 6.1. Comparación de tiempos de ejecución entre la propagación de una y dos GPU's con variaciones del dominio iguales para los tres Ejes

6.2. Prueba dos, transferencias constantes con un mayor dominio

El objetivo de esta prueba es verificar si manteniendo constantes las transferencias (dim X y dim Y iguales para cada ejecución) pero haciendo crecer el dominio en dirección Z, se presenta un mejor rendimiento de la estrategia descomposición de dominio. Para la segunda

prueba se varió solo la dimensión en Z y se mantuvieron contantes las dimensiones en X y en Y = 300. Igual que en la prueba anterior los tiempos se pudieron medir para hasta un dominio del doble de la propagación en una GPU, pero en este caso las mejoras en tiempos son valores alrededor de un 32 %.

Los resultados de la Tabla 6.2 se pueden visualizar en la Figura 6.2

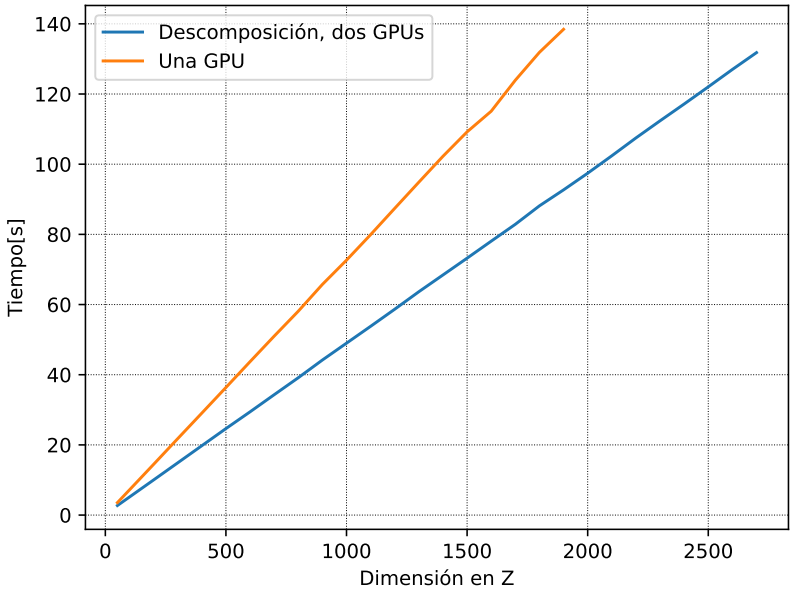

Figura 6.2. Comparación de tiempos de ejecución entre la propagación de una y dos GPU's con variación del Eje-z

6.3. Streams

Al cambiar la composición de los tres kernels a grid loop y ponerlos en modo concurrente, se vuelven hasta 7 veces más lentos como se observa en la Figura 6.3, en la cual se asignaron los recursos de los kernels de la siguiente manera $blockSize(4,4,4)$, $numBlock(4,4,4)$ y la dimensión del dominio es 100^3.

Tabla 6.2. Resultados 2, comparativa de tiempos entre la propagación en una y dos GPU's con variaciones en Eje-z

Dim_z	**Tiempo[s] usando dos GPU**	**Tiempo[s] usando una GPU**	**Comparación de tiempos[s]**	**% de mejora**
50	2,719	3,550	0,831	23,405
100	5,142	7,172	2,030	28,303
200	10,005	14,448	4,443	30,750
300	14,883	21,769	6,886	31,632
400	19,767	29,053	9,286	31,963
500	24,647	36,362	11,714	32,216
600	29,422	43,716	14,294	32,698
700	34,300	50,988	16,688	32,730
800	39,142	58,099	18,957	32,628
900	44,190	65,736	21,546	32,777
1000	48,991	72,648	23,657	32,564
1100	53,796	79,923	26,126	32,690
1200	58,671	87,421	28,750	32,887
1300	63,658	94,876	31,218	32,904
1400	68,433	102,258	33,825	33,078
1500	73,218	109,214	35,996	32,959
1600	78,064	115,102	37,038	32,178
1700	82,870	123,932	41,062	33,133
1800	88,131	131,880	43,749	33,173
1900	92,680	138,444	45,764	33,056
2000	97,430			
2100	102,391			
2200	107,478			
2300	112,357			
2400	117,150			
2500	122,038			
2600	126,999			
2700	131,795			

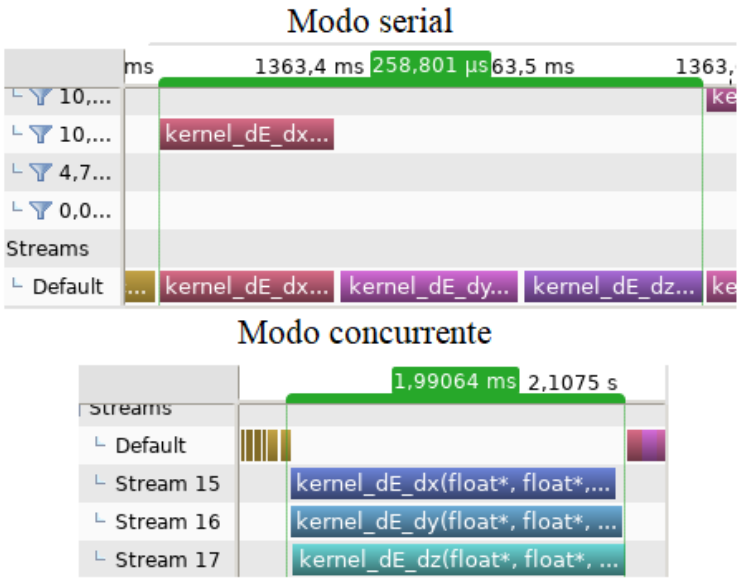

Figura 6.3. Kernels de las tres derivadas principales de primer orden en modo serial y modo concurrente

Al aumentar los recursos de los kernels, el número de bloques y dimensión se pierde la concurrencia como se evidencia en la Tabla 6.3, de la misma manera la implementación de los streams para los kernels de las derivadas de primero y segundo orden mostraron tiempos inferiores a la ejecución serial. El tiempo en modo serial para los kernels: $\frac{\partial E}{\partial x}$, $\frac{\partial E}{\partial y}$ y $\frac{\partial E}{\partial z}$ fue de $258.801\,[us]$ mientras que el mejor tiempo de la implementación de los streams conservando la concurrencia fue de $302.302\,[us]$ para un dominio 100^3.

Tabla 6.3. Resultados 3, tiempo de los kernels con streams de las derivadas de primer orden con variaciones en los recursos de los kernels

#	BlockZise	NumBlock	Tiempo [us]	Proceso
1	(4,4,4)	(4,4,4)	887,585	Concurrente
2	(16,16,4)	(4,4,4)	404,968	Concurrente
3	(16,4,4)	(16,16,16)	302,302	Concurrente
4	(16,16,16)	(4,4,4)	262,931	Serial
5	(16,16,16)	(16,4,4)	263,554	Serial
6	(4,4,4)	(16,16,16)	262,971	Serial

6.4. Transferencias asincrónicas

Esta implementación no es practica para dominios pequeños debido a que se aplica sobre la estrategia descomposición de dominio, en la cual la única transferencia asincrónica posible transfiere solo dos capas del dominio total, no obstante, entre mayor sea el dominio mejor es el resultado de la estrategia, como se evidencia en la Tabla comparativa 6.4, para un dominio de 100^3 y $1000n$ se observa una mejora del $3.7\,\%$, mientras que para un dominio de 300^3 y $1000n$ se observa una mejora del $10.17\,\%$.

Tabla 6.4. Resultados 4, Comparativa entre la descomposición de dominio con y sin transferencia asincrónica para dos dominios diferentes

Tamaño del dominio	Descomposición de dominio[s]	Descomposición de dominio con transferencia asíncrona[s]	Diferencia de tiempos [s]	% de mejora
100^3	1.993	1.9188	0.0743	3.726
300^3	17.7325	15.928	1.8045	10.1761

7. CONCLUSIONES

Este trabajo presenta la propagación de una onda electromagnética 3D con CPMLs y una descomposición de dominio con la que se dividió la propagación en dos GPU's. Se comprobó que la implementación en dos GPU's es hasta un $30\,\%$ más rápida y adicionalmente permite propagar la onda en dominios del doble del tamaño al que se puede utilizando una sola GPU, como se observa en la Tabla 6.1. Además, la implementación de las transferencias asincrónicas mostró mejoras adicionales a la descomposición de dominio de $3.7\,\%$ para un dominio 100^3 y $10.17\,\%$ para un dominio de 300^3 presentes en la Tabla 6.4. Por otra parte, no se presentaron cuellos de botella en las transferencias entre GPU's debido a que para mantener los dominios conectados solo son necesarias dos capas. Finalmente, los streams no muestran mejora en los tiempos de ejecución debido a que los kernels se aplicaron de forma cíclica, estos son de utilidad en casos donde el número de kernels paralelizables sea mayor.

BIBLIOGRAFIA

Baker, Gregory, Thomas Jordan y Jennifer Pardy. “An introduction to ground penetrating radar (GPR)”. En: vol. 432. Ene. de 2007, págs. 1-18. DOI: 10.1130/2007.2432(01) (vid. pág. 11).

Cheng, J., M. Grossman y T. McKercher. *Professional CUDA C Programming*. EBL-Schweitzer. Wiley, 2014 (vid. págs. 12, 25, 29, 33).

Houle, J.E. y D.M. Sullivan. *Electromagnetic Simulation Using the FDTD Method with Python*. Wiley, 2020 (vid. pág. 12).

Inan, U.S. y R.A. Marshall. *Numerical Electromagnetics: The FDTD Method*. Cambridge University Press, 2011 (vid. págs. 15, 16).

Liu, Yang y Mrinal K. Sen. “An implicit staggered-grid finite-difference method for seismic modelling”. En: *Geophysical Journal International* 179.1 (oct. de 2009). DOI: 10.1111/j.1365-246X.2009.04305.x. eprint: https://academic.oup.com/gji/article-pdf/179/1/459/5906542/179-1-459.pdf (vid. pág. 17).

Mao, Lei. *Cuda Stream*. 2022 (vid. pág. 29).

Pasalic, Damir y Ray McGarry. “Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations”. En: *SEG Technical Program Expanded Abstracts 2010*. DOI: 10.1190/1.3513453. eprint: https://library.seg.org/doi/pdf/10.1190/1.3513453 (vid. pág. 18).

Weiss, Robin M. y Jeffrey Shragge. “Solving 3D anisotropic elastic wave equations on parallel GPU devices”. En: *GEOPHYSICS* 78.2 (2013), F7-F15. DOI: 10.1190/geo2012-0063.1. eprint: https://doi.org/10.1190/geo2012-0063.1 (vid. pág. 11).

www.stepofweb.com, Dorin Grigoras. *Georadar de Penetración terrestre* (vid. pág. 17).