Solución Paralela en un GPU para la Ecuación de Advección-Difusión-Reacción mediante Diferencias Finitas Explícitas e Implícitas

Cesar Ivan Abrajan Barraza[a], Juan José Tapia Armenta[a]

[a]*Centro de Investigación y Desarrollo de Tecnología Digital del Instituto Politécnico Nacional*
Carr. Av. del Parque 1310 Mesa de Otay, Tijuana, B.C., México
cabrajan@citedi.mx, jtapiaa@ipn.mx

2013 Published by $\mathcal{DIFU}_{100}ci@$ *http://www2.uaz.edu.mx/web/www/publicaciones*
Selection and peer-review under responsibility of the Organizing Committee of the CICOMP-2013, www.cicomp.org

Resumen

En este trabajo se presenta la ecuación de Advección-Difusión-Reacción (ADR) que se paraleliza en una unidad de procesamiento grafico (GPU) para una, dos y tres dimensiones. Dicha ecuación se implementa mediante dos esquemas del método de finitas. El esquema explícito se implementa mediante la ecuación de recurrencia que se forma, sin embargo, tiene restricciones de estabilidad. Por otro lado, el esquema implícito forma un sistema de ecuaciones que se resuelve mediante el método iterativo de gradiente biconjugado en el GPU que obtiene ventaja de CUBLAS y CUSPARSE para resolver la matriz rala presente en dicho sistema de ecuaciones para cada dimensión. Finalmente se presentan los resultados donde se muestran el costo en tiempo de ejecución y convergencia que se obtiene al aplicar el gradiente biconjugado.

Palabras clave: Advección-Difusión-Reacción, Diferencias Finitas explícitas, Diferencias Finitas implícitas, GPU.

1. Introducción

La ecuación de Advección-Difusión-Reacción (ADR) es una ecuación diferencial de segundo orden que se usa ampliamente en el modelado matemático, su importancia radica en los tres fenómenos que agrupa. El término de advección indicado por los gradientes modela el desplazamiento de partículas a una cierta velocidad, especificada por su coeficiente. El término de difusión, representado con gradientes de segundo orden, modela el flujo de una propiedad física de regiones con concentración alta a regiones con concentración baja, hasta lograr un equilibrio. La rapidez del flujo está determinado por su coeficiente y el término de la reacción es una función que aporta o retira energía al sistema; es decir, representa la fuente

del sistema. En este trabajo se modela la ecuación ADR para estudiar la evolución de su solución, a partir de una solución inicial en el tiempo $t = 0$, con las condiciones de frontera apropiadas, de acuerdo al fenómeno que se desea estudiar.

La ecuación ADR se puede aplicar para resolver problemas en una variedad de disciplinas. En particular, en el área de ecología, permite hacer estudios de medio ambiente, entre los que destacan el estudio de la interacción entre especies, el deterioro de la calidad del aire en las grandes ciudades, y la simulación de daño ambiental en fugas de plataformas petroleras. En estas aplicaciones, el término que toma mayor relevancia es el de advección.

La ecuación de calor en los sistemas térmicos modela la energía calorífica que se propaga en diversos materiales, lo que permite analizar y aprovechar dichas propiedades para la transmisión o retención de la energía. De la misma manera, en las plantas nucleares usan esta ecuación para la simulación de enfriamiento de reactores, así como también la contaminación de materiales radiactivos. La ecuación ADR se usa en medicina para modelar el crecimiento de tumores cancerígenos [1]. En el campo de estudio de la química, el análisis de las reacciones entre substancias también puede modelarse mediante la ecuación ADR, siendo los patrones de Turing [2] los más conocidos.

Si bien existen casos donde la solución analítica o exacta es fácil de desarrollar, en la mayoría de los sistemas que son de importancia e impacto científico, es posible solo encontrar su aproximación mediante la aplicación de un método numérico que permita resolver el problema con un algoritmo computacional. Existen muchos métodos numéricos para resolver la ecuación ADR, entre los que destacan el método de diferencias finitas [3], método de elemento finito [4], método de diferencias finitas en el dominio del tiempo, métodos espectrales [5]. Un estudio detallado de la ecuación ADR se puede consultar en [6].

El modelado matemático tridimensional (3-D) mediante ecuaciones diferenciales es altamente demandante de recursos de cómputo. Hasta hace algunos años, estos problemas se resolvían en sistemas de cómputo muy costosos. Para abaratar los costos las grandes computadoras fueron sustituidas por clusters de computadoras y, en la actualidad, la opción más barata de usar cómputo de alto rendimiento es por medio de GPUs.

Es necesario también mencionar que no todos los métodos numéricos son ideales para implementarse en el GPU y es necesario saber distinguir las deficiencias de éstos [9]. En el presente trabajo se realiza el análisis numérico de la ecuación ADR mediante el método de diferencias finitas bajo dos esquemas, el explícito que es el más intuitivo al momento de paralelizar el sistema debido a que después del análisis numérico, la ecuación resultante en cada punto a calcular no presenta dependencia espacial en el mismo tiempo [10]. Con este esquema implementado en el GPU podemos encontrar trabajos de investigación con la ecuación de calor y de onda [11], en patrones de Turing [2], [12]-[14], análisis de sistemas térmicos mediante el acoplamiento de ecuaciones de Navier-Stokes. El otro esquema es el implícito, el cual después del análisis numérico resulta en un sistema de ecuaciones que de manera directa no se puede paralelizar (funciones kernel) debido a la dependencia de datos espaciales. Sin embargo, ésta misma situación abre la oportunidad de implementar nuevas formas de interacción con el GPU como lo son los métodos iterativos y el uso de aplicaciones de interfaces de muy alto nivel (API, del inglés *Application Programming Interface*), ya sean proporcionadas de manera oficial por los diseñadores del hardware como lo son CUBLAS, CUSPARSE, o bien por terceros.

2. Fenómenos de ADR

La ecuación de Advección-Difusión-Reacción [3] con dependencia temporal de una forma general se representa como:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u + \beta \nabla u + f(u) \tag{1}$$

donde el término de la izquierda representa el cambio de u en el tiempo. En el primer término de la derecha se representa la difusión con la segunda derivada espacial multiplicada por el coeficiente de difusión α. El segundo término representa la advección con la primera derivada respecto a la función u multiplicada por la constante del campo vectorial de velocidades β, y el tercer término representa una función de reacción.

Los fenómenos que se modelan con la ecuación ADR, pueden contener o no a todos los términos. En algunos casos, los valores que no aparecen se desprecian, tal es el caso de la ecuación del calor, donde el único término que aparece es el de difusión.

Para el caso específico de la ecuación de calor, la difusión representa el movimiento aleatorio de partículas que tienden a desplazarse de áreas con mayor energía calorífica a partes con menor intensidad. Dicho movimiento es directamente proporcional al coeficiente de difusividad del material por el cual se transporta dicha energía. Si al sistema se añade un líquido en reposo, éste tenderá a absorber o incorporar calor al sistema el cual dependerá de su temperatura. Pasado un tiempo,

tanto el líquido como el sistema tendrán la misma temperatura, debido a la difusión; por otro lado, si el líquido se encuentra en movimiento habrá un intercambio constante de energía calorífica además de un movimiento de arrastre debido a la advección.

Las condiciones de frontera intervienen directamente en el resultado de la ecuación ADR y pueden ser consideradas como funciones de reacción. En el caso de las condiciones de Dirichlet, presentan valores constantes que pueden o no cambiar en el tiempo y la ecuación que se modela tenderá a adoptar estos valores. Por otro lado, las condiciones de Neumann condicionan el flujo de la concentración que se analiza [17].

3. Aproximación numérica de la ecuación ADR mediante diferencias finitas

Para aproximar la solución de la ecuación ADR en el GPU se emplea el método de diferencias finitas, el cual consiste en remplazar las derivadas parciales por aproximaciones desarrolladas a partir de la serie de Taylor. Al ser una serie infinita, cada representación debe llevar un orden de truncamiento O (del inglés *big O*).

3.1. Aproximación numérica de la ecuación ADR con esquema explícito

Para aproximar la solución de la ecuación ADR en una dimensión, la Ec. (1) se representa como:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + \beta \frac{\partial u}{\partial x} + f(u) \tag{2}$$

El método de diferencias finitas bajo un esquema explícito, se inicia con una aproximación adelante en el tiempo (FT del inglés *Forward Time*) lo que resulta en:

$$\frac{\partial u}{\partial t} = \frac{u_j^{t+1} - u_j^t}{\triangle t} + O(\triangle t) \tag{3}$$

La segunda derivada espacial en x se calcula mediante el uso de diferencias finitas centradas (CS del inglés *Center Space*). El resultado que se obtiene es:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{j+1}^t - 2u_j^t + u_{j-1}^t}{\triangle x^2} + O(\triangle x^2) \tag{4}$$

Al igual que la segunda derivada, la primera derivada respecto a x se aproxima mediante CS obteniéndose:

$$\frac{\partial u}{\partial x} = \frac{u_{j+1}^t - u_{j-1}^t}{2 \triangle x} + O(\triangle x^2) \tag{5}$$

Al sustituir las Ecs. (3)-(5) en (2), se obtiene:

$$\begin{aligned} u^{t+1} &= u_j^t + \frac{\alpha \triangle t}{\triangle x^2}\left(u_{j+1}^t - 2u_j^t + u_{j-1}^t\right) \\ &+ \frac{\beta \triangle t}{2 \triangle x}\left(u_{j+1}^t - u_{j-1}^t\right) + \triangle t f\left(u_j^t\right) \end{aligned} \tag{6}$$

Al asignar $r = \alpha \triangle t / \triangle x^2$ y $s = \beta \triangle t / 2 \triangle x$ resulta:

$$\begin{aligned} u_j^{t+1} &= u_j^t + r\left(u_{j+1}^t - 2u_j^t + u_{j-1}^t\right) \\ &+ s\left(u_{j+1}^t - u_{j-1}^t\right) + \triangle t f\left(u_j^t\right) \end{aligned} \tag{7}$$

Al agrupar términos finalmente se obtiene:

$$u_j^{t+1} = a u_{j-1}^t + b u_j^t + c u_{j+1}^t + \triangle t f\left(u_j^t\right) \tag{8}$$

donde $a = r - s$, $b = 1 - 2r$ y $c = r + s$

En la Ec. (8) se aprecia como, independientemente del valor que contengan los coeficientes, la solución en el tiempo actual sólo dependerá de tres valores previamente calculados. Para el caso de dos dimensiones se sigue un proceso similar y la ecuación final es:

$$\begin{aligned} u_{i,j}^{t+1} &= a u_{i+1,j}^t + b u_{i,j+1}^t + c u_j^t + d u_{i-1,j}^t \\ &+ e u_{i,j-1}^t + \triangle t f\left(u_{i,j}^t\right) \end{aligned} \tag{9}$$

Finalmente para tres dimensiones la ecuación es:

$$\begin{aligned} u_{i,j,k}^{t+1} &= a u_{i,j,k+1}^t + b u_{i+1,j,k}^t + c u_{i,j+1,k}^t \\ &+ d u_{i,j,k}^t + e u_{i,j-1,k}^t + f u_{i-1,j,k}^t \\ &+ g u_{i,j,k-1}^t + \triangle t f\left(u_{i,j,k}^t\right) \end{aligned} \tag{10}$$

3.2. Aproximación numérica de la ecuación ADR con esquema implícito

El método de diferencias finitas tiene, entre sus variantes, el esquema implícito Crank-Nicolson (CN) [19]. Este esquema se utiliza en el modelado matemático para encontrar la solución numérica de ecuaciones diferenciales, siendo sus principales características que el error de aproximación es de segundo orden tanto en tiempo como en espacio. Es implícito e incondicionalmente estable.

La diferencia entre los esquemas CN y FTCS radica en la aproximación de diferencias finitas que se realiza en el tiempo, ya que CN utiliza los valores en el tiempo actual y el anterior, mientras que FTCS usa los valores en el tiempo anterior. Así también, CN realiza un promedio de los valores en el espacio entre los tiempos

anteriores y actuales, en tanto que FTCS solamente necesita valores previamente calculados.

Para aproximar la solución de la ecuación ADR en una dimensión mediante CN, de la Ec. [2], se aproxima la derivada temporal mediante diferencias finitas hacia atrás (BT del inglés *Back Time*) dada por:

$$\frac{\partial u}{\partial t} = \frac{u_j^t - u_j^{t-1}}{\triangle t} + O(\triangle t^2) \tag{11}$$

Para el término de difusión donde se involucra la segunda derivada, el esquema CN aproxima la solución mediante el promedio de diferencia finitas centradas en el espacio del tiempo anterior y el tiempo actual, lo que resulta en:

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2} &= \frac{1}{2 \triangle x^2}(u_{j-1}^t - 2u_j^t + u_{j+1}^t + u_{j-1}^{t-1} \\ &\quad -2u_j^{t-1} + u_{j+1}^{t-1}) + O(\triangle x^2)\end{aligned} \tag{12}$$

Para la primera derivada que aparece en el término de advección, el esquema CN nuevamente promedia mediante diferencias finitas centradas en el espacio del tiempo anterior y el actual, obteniéndose:

$$\frac{\partial u}{\partial x} = \frac{1}{4 \triangle x}\left(-u_{j-1}^t + u_{j+1}^t - u_{j-1}^{t-1} + u_{j+1}^{t-1}\right) + O(\triangle x^2) \tag{13}$$

Al sustituir las Ecs. [11]-[13] en Ec. [2] se obtiene el sistema de ecuaciones:

$$\begin{aligned}au_{j-1}^t + bu_j^t + cu_{j+1}^t = &- au_{j-1}^t + (1-2r)u_{j1}^t \\ &- cu_{j+1}^t + \triangle t f\left(u_j^{t-1}\right)\end{aligned} \tag{14}$$

con $a = s - r$, $b = 1 + 2r$ y $c = -s - r$. A su vez $r = \alpha \triangle t/2 \triangle x^2$ y $s = \beta \triangle t/4 \triangle x$. Las constantes a, b y c se guardarán en una matriz de coeficientes A del sistema de ecuaciones $Au = b$.

Para dos y tres dimensiones se siguen procedimientos similares, y únicamente crece el número de términos en cada renglón en el sistema de ecuaciones. La matriz de coeficientes A presenta la peculiaridad que, al tener pocos valores diferentes de cero, se les conoce como matriz rala. Así también al aumentar el número de dimensiones en la ecuación ADR se generará un matriz de coeficiente $N^2 \times N^2$ y de $N^3 \times N^3$ para dos y tres dimensiones respectivamente.

4. Implementación de la ecuación ADR en un GPU

La programación en paralelo se utiliza para resolver programas computacionalmente demandantes, para lo cual se utilizan diferentes paradigmas de tecnologías de software que permiten expresar algoritmos paralelos para implantar aplicaciones en diferentes arquitecturas [8]. En un modelo de programación se pueden incluir distintas áreas como lo son los diferentes lenguajes de programación, los compiladores, las interfases para el desarrollo de aplicaciones, las bibliotecas de funciones, sistemas de comunicación, dispositivos de entrada y salida [7].

El principal requisito para paralelizar un problema es que no exista dependencia de datos. Esto es, que permita la descomposición del problema original en partes de tal manera que puedan ejecutarse de manera simultánea sin afectar el resultado final. Otros aspectos adicionales a considerar son asignación de tareas y la comunicación entre los procesos.

Una ventaja de utilizar plataformas en paralelo es que es posible eliminar la limitación física de la computación secuencial donde la rapidez del programa es directamente proporcional a la frecuencia del reloj. Sin embargo, más frecuencia también significa más consumo de energía lo que conlleva a saturar el sistema [1]. Ante este inconveniente, seguir con el aumento de frecuencia no es una opción y, por lo tanto, los problemas con complejidad elevada o que requieren una solución en tiempo real se ven comprometidos. En este trabajo se presenta un problema que, además de tener dependencia temporal, que habrá de resolverse un cierto número de intervalos en el tiempo, tiene la peculiaridad de que al aumentar el número de dimensiones espaciales el costo computacional se incrementa de manera drástica.

CUDA (del inglés *Compute Unified Device Architecture*), es un conjunto de tecnologías tanto en hardware como en software que proporcionan a desarrolladores los recursos necesarios para la programación de la unidad de procesamiento gráfico. Desde el punto de vista del software, CUDA es un conjunto de funciones que pueden tratarse como extensiones de lenguajes de programación tales como C, C++, Fortran, Python, entre otros [20].

En este trabajo se utiliza CUDA C para implementar los algoritmos que resuelven la ecuación ADR. De manera general, se necesitan de cuatro elementos para desarrollar programas en un GPU [7]:

1. Un GPU que soporte CUDA.
2. Un controlador de recursos para el GPU.
3. Un conjunto de herramientas de desarrollo de CUDA.

Tabla 1. Tipos de memoria en el GPU

Memoria	Cache	Acceso	Accesibilidad
Registros	-	R/W	Hilo
Local	-	R/W	Hilo
Compartida	-	R/W	Bloque
Constante	Sí	R	Hilos + CPU
Textura	Sí	R	Hilos + CPU
Global	Sí	R/W	Hilos + CPU

4. Un compilador de lenguaje C.

Al ser el GPU un procesador auxiliar al momento de ejecutar un programa, el hilo principal siempre estará en el CPU y se ejecutará de manera secuencial. Cuando se inicializa una subrutina que se habrá de ejecutar en el GPU mediante una función kernel, la ejecución se mueve al GPU, donde un determinado número de hilos son ejecutados concurrentemente para realizar el cálculo de las operaciones en paralelo. En el momento de la ejecución en el GPU se ejecutan una gran cantidad de hilos. Estos hilos son agrupados en bloques, que a su vez son agrupados en mallas. Cada hilo tiene su propia memoria, así como una memoria compartida visible para el resto de los hilos en el mismo bloque. Existe, además, la memoria principal o global la cual es visible para todos los hilos y dos memorias tipo cache; es decir, memorias de muy rápido acceso pero que están sumamente limitadas en capacidad como son la memoria de textura y la memoria constante ambas únicamente de lectura. La tabla 1 muestra un resumen de las memorias en el dispositivo y sus características principales; sin embargo, cualquier memoria que puede ser modificada por el CPU requiere un copiado de memoria.

4.1. Algoritmo para el esquema FTCS

El algoritmo 1 mostrado en la Tabla 2 se utiliza en la implementación de la aproximación de la ecuación ADR en una, dos y tres dimensiones mediante el esquema de diferencias finitas explícitas. Se parte de la reservación de la memoria tanto en el GPU como en el CPU, después se utilizan las condiciones de la ecuación diferencial para establecer la función inicial, luego se adecúa la función solución a un vector y se copia a la memoria en el GPU, en cada iteración se inicializa un kernel que valida la convergencia de la solución por medio de otro kernel que evalúa el error de norma L2 al término de la iteración. Una vez obtenida la solución se copian los resultados al CPU, y finalmente se libera la memoria tanto en CPU como GPU.

La implementación del algoritmo 1 se realiza de una manera muy intuitiva debido a que no se presenta una dependencia de datos. Los programas son divididos en kernels, los cuales son compilados y ejecutados en el GPU de manera paralela y dependerá de la arquitectura del hardware que se utilice. Kernel es la porción de cómputo intensivo que mediante funciones en C se ejecutan N veces en N diferentes hilos [20].

Tabla 2. **Algoritmo 1**. Algoritmo para la ecuación ADR mediante FTCS en un GPU.

```
reservar memoria en el GPU.
reservar memoria en el CPU.
condiciones iniciales.
adecuar la función solución a un vector.
copiar memoria del CPU al GPU.
for (t = 1:N) do
   inicializar kernel.
   ejecutar kernel norma L2
end for
copiar solución del GPU al CPU.
resultados.
liberar memoria.
```

Tabla 3. **Algoritmo 2**. Algoritmo para la ecuación ADR mediante CN en el GPU.

```
reservar memoria en el GPU.
reservar memoria en el CPU.
condiciones iniciales.
formar matriz rala.
inicializar CUBLAs.
inicializar CUSPARSE.
for (t=1:N) do
   ejecutar BiCG.
   ejecutar kernel norma L2.
   actualizar solución con llamado de un kernel.
end for
copiar solución del GPU al CPU.
resultados.
liberar memoria.
```

4.2. Aproximación numérica de la ecuación ADR con esquema implícito

El algoritmo 2 mostrado en la Tabla 3 se utiliza en la implementación de la aproximación de la ecuación ADR en una, dos y tres dimensiones mediante el esquema de diferencias finitas explícitas. Se parte de la reservación de la memoria tanto en el GPU como en el CPU, después se utilizan las condiciones de la ecuación diferencial para establecer la función inicial, luego se adecúa la función solución a un vector y se copia a la memoria en el GPU. En cada iteración se inicializa un kernel que valida la convergencia de la solución por medio de otro kernel que evalúa el error de norma L2 al término de la iteración. Una vez obtenida la solución se copian los resultados al CPU, y finalmente se libera la memoria tanto en CPU como GPU.

En el algoritmo 2, aparecen nuevos términos como lo son CUBLAS, CUSPARSE, BiCG así como la matriz de

coeficientes A que resulta un caso de estudio [21] debido a que contiene muy pocos valores diferentes de cero. Estas matrices se conocen como matrices ralas que pueden formarse de manera aleatoria [22] y pueden ser tanto reales como complejas; así también las matrices ralas son predefinidas [21], [23]. Existen universidades que proveen base de datos con matrices ralas con características únicas en cuanto a su representación por los patrones que pueden formar, y dichos patrones resultan al aplicar diferentes métodos numéricos en la resolución de fenómenos, aunque no se descarta la creación de matrices ralas de manera artificial.

Los sistemas de ecuaciones $Au = b$ se pueden resolver mediante métodos directos e iterativos. De los primeros resaltan los métodos de eliminación directa como Gauss y sus variantes o bien descomposición LU donde L es una matriz triangular inferior y U es una matriz triangular superior. Para el caso de una implementación en el GPU se realiza mediante los métodos iterativos como el del gradiente biconjugado; por tal motivo, para implementar en paralelo la solución de un sistema de ecuaciones, se plantea como un problema de optimización puesto que se tiene que minimizar el residuo r en $Au - b = r$ [24].

Para realizar las operaciones necesarias en el algoritmo 2 al momento de realizar el gradiente biconjugado se realiza un cambio en el formato de la matriz rala. Se descompone en vectores, el caso más sencillo es guardar cada una de las posiciones con su respectivo valor. Este formato es conocido como completo o crudo [25], y es el menos recomendado debido a que los tres vectores serán muy grandes llegando incluso a superar el espacio de memoria con el que se cuenta. Por tal motivo, en este trabajo no se realizarán operaciones con este formato en la solución de la ecuación ADR.

Para la implementación de la ecuación ADR con esquema implícito se requiere el uso de CUBLAS que es una API que proporciona las subrutinas para las operaciones básicas de álgebra lineal, además de que permite el acceso, manejo y monitoreo de recursos en el GPU de una manera más eficiente. Las operaciones pueden ser de tipo flotante, dobles, complejas flotantes y complejas dobles, las cuales dependen de la capacidad de cómputo en el GPU. En este trabajo CUBLAS se utilizó para implementar el gradiente biconjugado en las operaciones con vectores densos o completos como el producto punto. Las rutinas se clasifican en:

Nivel 1 (BLAS1). Funciones para realizar operaciones tipo escalar-vector.

Nivel 2 (BLAS2). Funciones para realizar operaciones tipo matriz-vector.

Tabla 4. Comparación de los métodos explícito e implícito en una dimensión.

N	Explícito FTCS 1D		Implícito CN 1D	
	Duración	Iteración	Duración	Iteración
32	1.5	19799	5.9	19781
64	1.6	20133	6.3	20135
128	1.6	20483	6.4	20485
256	1.6	20831	6.6	20834

Nivel 3 (BLAS3). Funciones para realizar operaciones tipo matriz-matriz.

CUSPARSE es la API mediante la cual se realizan las operaciones con matrices ralas. Al igual que en CUBLAS, todas las instrucciones se invocan en el CPU y las ejecuta el GPU. CUSPARSE exige que la matriz sea convertida previamente a un formato de matriz rala y provee instrucciones para convertir de manera automática entre los formatos. En este trabajo se utilizó en todas las operaciones donde los vectores o matrices no fueran densos. Al igual que CUBLAS se pueden realizar operaciones del tipo flotante, doble, complejo flotante y complejo doble. Las rutinas se clasifican en:

Nivel 1. Operaciones entre vectores en formato de matriz rala y vectores en formato denso.

Nivel 2. Operaciones entre una matriz en formato de matriz rala y vector en formato denso.

Nivel 3. Operaciones entre una matriz y un conjunto de vectores, ambos en formato de matriz rala.

Conversión. Operaciones para convertir entre los diferentes formatos de matrices ralas.

5. Simulación

En la Tabla 4 se muestra los valores en tiempo que se necesitan para resolver la ecuación ADR en un procesador gráfico al variar la cantidad de puntos en la malla unidimensional. Se usó la función inicial $u = \text{sen}(\pi x/L)$, con condiciones en la frontera del tipo Dirichlet homogéneas. Para los diferentes valores en N se utilizó $\alpha = 10^{-5}$ y un coeficiente $\beta = 10^{-7}$. El fenómeno de reacción está dado por la función $u(x,t)$, y el incremento en el tiempo fue de 10^{-3}. La longitud de prueba se fijó a $L = 1$. Para asegurar la convergencia de la función a 10^{-8} se usó la norma L2. En las Tablas 4, 5 y 6 la columna duración indica el tiempo de ejecución en milisegundos.

En la Tabla 5 se presentan los valores encontrados tanto en tiempo como en número de iteraciones para converger al equilibrio de la ecuación ADR en dos dimensiones al usar el método de diferencias finitas explícitas como implícitas. Para llevar a cabo esta simulación

Tabla 5. Comparación de los métodos explícito e implícito en dos dimensiones.

N	Explícito FTCS 2D		Implícito CN 2D	
	Duración	Iteración	Duración	Iteración
32^2	1.8	21389	7.2	21392
64^2	1.8	22122	8.7	22126
128^2	3.6	23532	12.6	23529

Tabla 6. Comparación de los métodos explícito e implícito en tres dimensiones.

N	Explícito FTCS 3D		Implícito CN 3D	
	Duración	Iteración	Duración	Iteración
8^3	2.4	21833	8.00	21892
16^3	2.41	22798	74.10	27533
32^3	14.29	42303	1612.80	51759

se utilizó la función inicial de $u = \text{sen}(\pi x/L_x)+\text{sen}(\pi y/L_y)$. Los coeficientes de difusión fueron $\alpha_x = \alpha_y = 10^{-5}$, y los coeficientes de advección $\beta_x = \beta_y = 10^{-7}$. La función de reacción se define como u, el incremento en el tiempo fue de 10^{-3}, para asegurar la convergencia para 10^{-8} se utilizó la norma L2. La longitud para cada dimensión fue de $L_x = L_y = 1$, por último se establecieron condiciones de frontera de Dirichlet homogéneas.

De la Tabla 5 se puede concluir que la ecuación de ADR para dos dimensiones muestra que los tiempos de ejecución, en el método explícito, se ven afectados un mínimo respecto a los tiempos obtenidos en una dimensión. Por el contrario, el método implícito presenta una tendencia a la alza, para las iteraciones de convergencia se presentan cambios mínimos o no significativos.

En la Tabla 6 se presentan los resultados de la simulación en un espacio tridimensional de la ecuación ADR. Se usó la función inicial de $u = \text{sen}(\pi x/L_x) + \text{sen}(\pi y/L_y) + \text{sen}(\pi z/L_z)$. Los coeficientes de difusión fueron $\alpha_x = \alpha_y = \alpha_z = 10^5$, y los coeficientes de advección fueron $\beta_x = \beta_y = \beta_z = 10^{-7}$. La función de reacción se define como u, el incremento en el tiempo fue de 10^{-3}, y para asegurar la convergencia de 10^{-8} se utilizó la norma L2. La longitud para cada dimensión fue de $L_x = L_y = L_z = 1$. Por último se establecieron condiciones de frontera de Dirichlet homogéneas.

De la Tabla 6 se aprecia la convergencia de la aproximación mediante la norma L_2 con condiciones de frontera de Dirichlet. En este caso en particular donde todas las dimensiones presentan condiciones de igualdad, el método explícito destaca tanto por el tiempo de ejecución como de iteración en convergencia. Al aumentar el número de dimensiones el costo por iteración en el método explícito se ve mínimamente afectado, puesto que solo aumenta el tamaño del vector que se envía al procesador gráfico. Por otro lado, el método implícito necesita resolver en cada iteración temporal una serie de iteraciones mediante el gradiente conjugado, lo cual tiene que afectar directamente el tiempo de ejecución y su tiempo de convergencia necesita de la solución de dos normas L_2: una en la iteración en el tiempo y una en cada iteración dentro del gradiente biconjugado, lo que se ve reflejado en la simulación con 32^3 elementos.

6. Conclusiones y Trabajo Futuro

En este trabajo, se resuelve la ecuación de Advección-Difusión-Reacción con el método de diferencias finitas en un procesador gráfico. El esquema explícito se paraleliza de una manera muy intuitiva al resolver una ecuación de recurrencia en la que cada hilo de ejecución resuelve una incógnita. El esquema implícito se paraleliza al plantear un sistema de ecuaciones como un problema de optimización y se resuelve en el GPU mediante las bibliotecas de funciones CUBLAS y CUSPARSE. Para medir los tiempos de ejecución y la convergencia de los esquemas en las mismas circunstancias se emplearon parámetros similares. En la comparación para una y dos dimensiones los tiempos de ejecución del esquema explícito muestra ventaja; sin embargo, la convergencia en iteración es prácticamente la misma. Para la ecuación ADR evaluada en tres dimensiones, el costo computación aumenta dramáticamente, especialmente en el método implícito, debido a que la matriz que se tiene que resolver es de $N^3 \times N^3$ y esto se ve reflejado en el tiempo de ejecución. Además, resolver el gradiente biconjugado envuelve un error al calcular el residuo en cada intervalo de tiempo y, por lo tanto, la convergencia para tres dimensiones requiere más iteraciones. Para mejorar el método implícito es necesario mejorar el método de optimización.

Como trabajo a futuro se propone:

- Mejorar el algoritmo del gradiente biconjugado mediante el uso de matrices de pre-acondicionamiento para disminuir el tiempo de convergencia para cada iteración en el tiempo. También, usar métodos de optimización alternativos como son el método del residuo generalizado.

- Cuando la reacción domina a la difusión se tienen cambios bruscos en la solución en una región muy pequeña de la solución. Por lo que se requiere adaptación de malla. Como trabajo futuro se propone usar métodos de adaptación de malla en el GPU.

- Explotar las capacidades de la memoria de solo lectura o memoria de textura para mejorar el desempeño en la implementación en el esquema explícito.

Por otro lado, el uso de la memoria compartida podría ser una alternativa viable.

- Implementar las ecuaciones de Navier-Stokes para modelar fluidos.
- Implementar la ecuación ADR en el GPU mediante el uso de métodos numéricos alternativos como el método de elemento finito y métodos espectrales.
- Para el caso de sistemas que requieren del esquema implícito, el cálculo en tres dimensiones con una gran cantidad de datos, la memoria del GPU no es suficiente. Para este caso, se recomienda el uso de un clúster de GPUs.

Agradecimientos

Se agradece el apoyo recibido por parte del proyecto SIP-20130489.

Referencias

[1] X. Chen, R. Summers, and J. Yao, "Kidney tumor growth prediction by coupling reaction diffusion and biomechanical model," *IEEE Trans. Biomed. Eng.*, Vol. 60, No. 1, pp. 169-173, 2013.

[2] J. Villagrana Mancilla. L. Yamamoto, D. Miorandi, P. Collet, and W. Banzhaf, "Recovery properties of distributed cluster head using reaction-diffusion," *Swarm Intell.*, Vol 5, No. 3-4, pp 225-255. 2011.

[3] A. R. Sanderson, M. D. Meyer, R. M. Kirby, and C. R. Johnson, "A framework for exploring numerical solutions of advection-reaction-diffusion equations using a GPU-based approach," *Comput. Vis. Sci.*, Vol. 12, No. 4, pp. 155-170, 2009.

[4] M. Bause and K. Schwegler, "High order finite element approximation of systems of convection-diffusion-reaction equations with small diffusion," *J. Comput. Appl. Math.*, Vol. 246, pp. 52-64, July 2013.

[5] F. Shakeri and M. Dehghan, "The finite volume spectral element method to solve Turing models in the biological pattern formation," *Comput. Math. Appl.*, Vol. 62, No. 12, pp. 4322-4336, 2011.

[6] W. Hundsdorfer and J. Verwer, *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Springer, 2003.

[7] J. Sanders and E. Kandrot, *CUDA by example: An introduction to General-Purpose GPU Programming*. Addison-Wesley, 2010.

[8] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, eds., *Sourcebook of parallel computing*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[9] M. H. Holmes, *Introduction to Numerical Methods in Differential Equations*. Springer, 2007.

[10] A. Bourchtein and L. Bourchtein, "Explicit finite difference scheme with extended stability for advection equations," *J. Comput. Appl. Math.*, Vol. 236, pp. 3591-3604, September 2012.

[11] C. A. Nahas, C. V. Krishnamurthy, K. Balasubramaniam, and P. Rajagopal, "Graphics processing unit based computation for NDE applications," *AIP Conf. Proc.*, Vol. 1430, pp. 1998-2005, 2012.

[12] R. Acar, "An advection reaction model for flow visualization," *Pacific Visualization Symposium (PacificVis), 2010 IEEE*, pp. 137-144, Mar 2010.

[13] D. G. Alvarado, C. Galeano, and J. Mantilla, "Turing pattern formation for reaction convection diffusion systems in fixed domains submitted to toroidal velocity fields," *App. Math. Model.*, Vol. 35, pp. 4913-4925, 2011.

[14] F. Molnár, F. Izsák, R. Mészáros, and I. Lagzi, "Simulation of reaction diffusion processes in three dimensions using CUDA," *Chemometr. Intell. Lab.*, Vol. 108, pp. 76-85, 2011.

[15] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007.

[16] D. M. Causon, *Introductory Finite Difference Methods for PDEs*. Ventus Publishing ApS, 2010.

[17] A. Quarteroni, *Numerical Models for Differential Problems*. Springer, 2009.

[18] W. E. Boyce and R. C. DiPrima, *Elementary Differential Equations and Boundary Values Problems*. Laurie Rosatone, ninth ed., 2009.

[19] J. Crank and P. Nicolson, "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type," *Adv. Comput. Math.*, Vol. 6, pp. 207-226, 1996.

[20] D. B. Kirk, *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers, Elsevier, 2010.

[21] A. Ahamed and F. Magoules, "Iterative methods for sparse linear systems on graphics processing unit," *IEEE 14th International Conference on High Performance Computing and Communication*, Liverpool, UK, pp. 836-842, june 2012.

[22] G. Ortega, E. Garzon, F. Vazquez, and I. Garcia, "The biconjugate gradient method on GPUs," *J. Supercomput*, Vol. 64, pp. 49-58, 2013.

[23] R. Li and Y. Saad, "GPU-accelerated preconditioned iterative linear solvers," *J. Supercomput*, Vol. 63, No. 2, pp. 443-466, 2013.

[24] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.

[25] NVIDIA, CUBLAS Library. NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2012.

[26] NVIDIA, CUSPARSE Library. NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2012.