Semi-Lagrangian Implementation Method using GPU

Yensy Gómez, Juan Hincapié, John Osorio, Julio Vargas, *Fellow, Sirius HPC , UTP*

***Abstract*—Nowadays plasma has become of great importance in both scientific and industrial areas since nanostructures, used for creating small computational processing devices, are produced by pulse lasers. The Simulation of phenomena such as plasma consumes a big amount of CPUs execution time. This paper intends to suggest a parallel solution of the Semi-Lagrangian using GPUs to solve the Vlasov-Poisson equations in 1D, as a different simulation alternative to PIC (Particle in Cell). To make this process a CUDA was used as a programing framework and an nVidia GTX 580 graphic card as a parallel processing element. The result of the parallelization process allowed accelerating the execution time of 3.7 times faster in comparison to the sequential algorithm.**

***Index Terms*—GPUs, Vlasov-Poisson, Semi-Lagrangian Method, HPC.**

I. Introducción

Las ecuaciones de Vlasov-Poisson juegan un papel importante en la física del plasma, debido a que describen el movimiento de las partículas en este estado. La solución numérica de las ecuaciones depende de tres variables tiempo t, posición x y velocidad v, que se soluciona a partir del uso de métodos como PIC [2]. Este método permite tener una solución satisfactoria con pocas partículas. Sin embargo, con un alto número de partículas se obtiene mucho ruido y se pierde precisión, debido a que agrupa las partículas a analizar en un número finito de "súper partículas". Por el contrario, el método semi-lagrangiano tiene una mejor precisión a nivel físico ya que tiene en cuenta todas las partículas durante el análisis [3]. Esto último significa también que requiere muchos más recursos computacionales para su implementación.

Aunque métodos como PIC y el método semi-lagrangiano se utilizan a menudo para estudios físicos, el consumo en tiempo de computador es bastante significativo debido a la gran cantidad de cálculos que se debe realizar para estudiar cada una de las partículas que se encuentran en el espacio de fase. Por lo tanto, aprovechando la gran escalabilidad y la poca dependencia de datos del método semi-lagrangiano se puede pensar en dispositivos que sean altamente paralelos como las GPUs (Unidades de procesamiento Gráfico) [4] que permitan dar una solución en menor tiempo comparados con soluciones secuenciales.

Estos tipos de algoritmos encajan muy bien con el uso de frameworks de programación como CUDA [4], ya que no solo mejoran la velocidad de ejecución de los algoritmos, si no también la precisión de las simulaciones. En este artículo se presenta entonces dicha solución, para lo cual en el primer capítulo se introduce el modelo matemático de las ecuaciones que modelan el comportamiento de las partículas en un plasma. Después se realiza una breve introducción del método semi-lagrangiano. Luego se presenta el algoritmo secuencial de la solución de las ecuaciones y la paralelización de la misma. Por último se realiza una comparativa de los tiempos ejecutados en CPU y GPU.

978-1-4799-6717-9/14/$31.00 © 2014 IEEE

II. Modelo Matemático de las Ecuaciones de Vlasov-Poisson

En esta sección se presenta una breve descripción en 1D del modelo físico de plasma donde se evalúa la evolución temporal de las partículas cargadas en un plasma donde se usará el método semi-lagrangiano.

A. Modelo de las Ecuaciones de Vlasov-Poisson

Se considera el modelo clásico de las ecuaciones de Vlasov-Poisson en $1D$, donde la función de distribución de las partículas esta dada por $f(t, x, v)$, la cual es dependiente de las variables posición $x[0, L]$ donde $L > 0$, a su vez L es el tamaño del dominio, la velocidad $v \ln[R]$ y el tiempo es $t \leq 0$, por lo tanto las ecuaciones de Vlasov se pueden expresar como:

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} + E\frac{\partial f}{\partial v} = 0$$

Para solucionar la ecuación de campo eléctrico $E = E(x, t)$, se utiliza la ecuación de Poisson que está dada por:

$$-\epsilon_0 \bigtriangledown^2 \phi = \rho(x, v) = q \int f(t, x, v), E(x, t) = - \bigtriangledown \phi$$

Donde ρ es la densidad de carga de las partículas, q es la carga de la partícula, ϵ_0 es la permitividad eléctrica y ϕ es el potencial electroestático. Para solucionar este sistema de ecuaciones se necesitan de métodos numéricos que sea capaces de predecir la evolución en el tiempo a nivel del movimiento cinético.

III. Método Semi-Lagrangiano

El método semi-lagrangiano es un tipo de método de advección numérico que se utiliza a menudo en la predicción del clima. Suponga que se desea modelar el flujo de un fluido [5] [6], para esto se puede elegir un método que permita simular el movimiento del agua.

Suponiendo que el método que se tomó para solucionar el problema es el semi-lagrangiano entonces el flujo se analizaría desde dos puntos de vista: El primero es el punto de vista absoluto donde se usa el esquema euleriano; este método crea una malla computacional, donde cada punto de la malla tiene asociado valores de las variables resolviendo el problema desde un sistema de referencia fijo. El segundo punto de vista es el relativo que es un marco de referencia que se mueve con el flujo; a este esquema se le conoce como el método lagrangiano, lo que hace este esquema es que la malla computacional se mueve con las partículas de flujo y mapea la trayectoria de manera individual.

El método lagrangiano es estable durante pasos de tiempo amplios, pero las partículas pueden extenderse a lo largo de una gran superficie, lo que ocasiona que el método se torne complejo en un área pequeña, por lo que es difícil estimar los gradientes, lo que se traduce en un modelo poco preciso.

El método semi-lagrangiano es una combinación de los anteriores esquemas, en los que procura tener las mejores propiedades de los dos métodos (el método euleriano y lagrangiano) es decir, manteniendo la red-computacional fija al marco euleriano, pero sin olvidar la estabilidad de los pasos de tiempo, en cuanto el plano Lagrangiano.

El principio básico del método semi-lagrangiano consiste en utilizar un conjunto diferente de las partículas para cada paso de tiempo, y el uso de los valores en los puntos de la malla del paso de tiempo anterior a la aproximación de los valores de los puntos de malla para cada nuevo paso de tiempo. Esto permite que con el método semi-lagrangiano se puedan resolver muchas ecuaciones de advección como lo son las ecuaciones de Vlasov-Poisson, por lo tanto se considera una malla de estudio (espacio de fase), con N_x, que es el número de puntos en la dirección x es decir $[0, L_x]$, por lo tanto se define lo siguiente:

$$\delta x = \frac{L_x}{N_x}, x_i = i\delta x_i]$$

para $i = 0, ..., N_x$, un punto importante que presenta este método es la definición de la función de distribución a través de spline cúbicos [7]:

$$f(t, x) = \sum_{k}^{N} \omega_k^n S(x_i^n - x_k^n)$$

Esto corresponde a la solución de las características en un tiempo t (en un sistema de $1D$), que se encuentra en un punto de la malla; los coeficientes del spline cúbico están definidos por las siguientes condiciones de interpolación: En

$$6S(x) = \begin{cases} (2 - |x|)^3 & \text{if } 1 \le |x| \le 2, \\ 4 - 6x^2 + 3|x|^3 & \text{if } 0 \le |x| \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

Fig. 1: spline.

la expresión el peso W, esta asociado a la partícula localizada en un punto de la malla en un tiempo t^n, esto corresponde a los coeficientes del spline cubico que está determinado por las siguientes condiciones:

$$f(t^n, x_i) = \sum_{k}^{N} \omega_k^n S(x_i^n - X(t^n, X_k))$$

Las condiciones de borde para solucionar este sistema se ven en [8] donde se utilizan métodos de interpolación como spline cúbicos o hermite spline.

IV. Implementación Secuencial

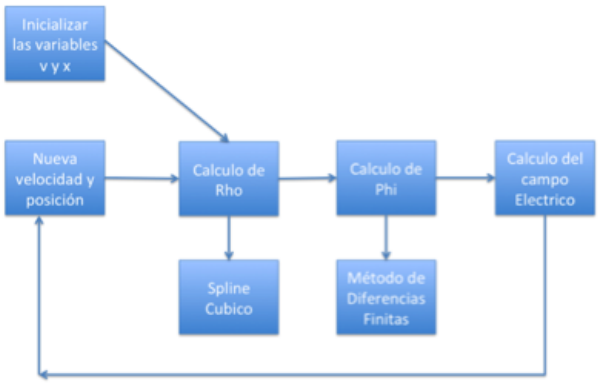

Fig. 2: Modelo Secuencial.

Considerando un sistema lineal sin colisiones, se analiza el comportamiento de las partículas dentro de un plasma en un espacio de fase unidimensional, explorando el método Semi-lagrangiano. Es importante aclarar que la ejecución del algoritmo secuencial se hizo en unidades de propósito general (CPU).

1) Paso 0: En este punto se inicializan las variables de la velocidad v y la posición x para resolver las ecuaciones de Vlasov-Poisson con respecto a la función de distribución de las partículas; dependiendo del tipo de plasma que se esté estudiando se debe tomar una determinada función de distribución, pero con fines de solucionar las ecuaciones para obtener los tiempos de ejecución, se toma la siguiente función de distribución que se encuentra de manera explícita en [9]. Por lo tanto: Se inicializa

$$f_i^0 = f_0(x_i, v_i)$$

$$f_0(x_i, v_i) = \frac{1}{\sqrt{2\pi}} e^{\frac{-v_i^2}{2}} (1 + \alpha \cos(k x_i)); (x_i, v_i) \epsilon (0, \frac{2\pi}{k})$$

2) Paso 1: En el paso 1, se debe obtener la densidad de carga Rho, esto se hace encontrando el valor de W en la siguiente ecuación.

$$f_0(x_i, v_i) = \sum_k^N \omega_k^0 S(x_i^0 - X(t^0, X_k))$$

esta expresión puede escribirse como:

$$AW^n = F^n$$

$$F^n = (F^{n^1}(x_0), ..., F^n(x_n), F^{n^1}(x_n))^T$$

$$W^n = (\omega_{-1}^n, \omega_0^n, \omega_1^n, ..., \omega_N^n, \omega_{N+1}^n)^T$$

donde:

$$A = 1/6 \begin{pmatrix} -3/h & 0 & 3/h & 0 & \cdots & 0 \\ 1 & 4 & 1/h & 0 & \cdots & \vdots \\ 0 & 1 & 4/h & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & 0 & 1 & 4 & 1 \\ \vdots & \vdots & 0 & -3/h & 0 & 3/h \end{pmatrix}$$

3) Paso 2: Al obtener los pesos se obtiene la densidad de carga de las partículas en un espacio de fase para luego calcular lo siguiente:

4) Paso 2.1:

$$\forall_k v_k^{n+1/2} = \frac{\triangle t}{2} (E(X_k^n, t^n)) + v_k^n$$

5) Paso 2.2:

$$\forall_k x_k^{n+1/2} = \triangle t v_k^{n+1/2} + v_k^n$$

6) Paso 2.3:

$$\rho(x_i, t^n) = \sum_k^N \omega_k^0 S(x_i^0 - X(t^0, X_k))$$

$$\bigtriangledown^2 \phi = \frac{\rho}{\epsilon_0}$$

Al tener la solución de $\rho(x_i, t^n)$ se puede calcular el potencial electroestático para luego calcular el campo eléctrico. Como en este caso se esta entrando en un estado inicial, se debe calcular el campo eléctrico teniendo en cuenta los aspectos físico matemáticos dados por [9], donde :

$$\rho(x, t = 0) = 0.0015 sech \frac{x}{2}$$

Obteniendo este resultado se entra a solucionar la siguiente matriz que resulta de aplicar el método de diferencias finitas para obtener el Potencial Electroestático, con el cual se obtiene el campo eléctrico E.

$$\begin{pmatrix} b_{r_2} & c_{r_2} & 0 & & \\ a_{r_3} & b_{r_3} & c_{r_3} & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & a_{r_{N_r-2}} & b_{r_{N_r-2}} & c_{r_{N_r-2}} \\ & & 0 & a_{r_{N_r-1}} & b_{r_{N_r-1}} \end{pmatrix} \begin{pmatrix} \Phi_2^{m,n} \\ \Phi_3^{m,n} \\ \vdots \\ \Phi_{N_r-2}^{m,n} \\ \Phi_{N_r-1}^{m,n} \end{pmatrix} = \begin{pmatrix} \rho_2^{m,n} - a_{r_2} \Phi_1^{m,n} \\ \rho_3^{m,n} \\ \vdots \\ \rho_{N_r-2}^{m,n} \\ \rho_{N_r-1}^{m,n} - c_{r_{N_r-1}} \Phi_{N_r}^{m,n} \end{pmatrix}$$

Fig. 3: Método de Diferencias Finitas.

Con estos cálculos ya es posible decir que se puede hallar la nueva velocidad en un punto de la malla, y al obtener la velocidad se obtiene la posición de la partícula.

7) Paso 2.4:

$$\forall_k v_k^{n+1} = \frac{\triangle t}{2} (E(X_k^{n+1}, t^{n+1})) + v_k^{n+1/2}$$

8) Paso 3:

$$f_i^{n+1} = \sum_k^N \omega_k^n S(x_i^{n+1} - X_k^{n+1})$$

9) Paso 4: Se calcula el coeficiente del spline cúbico para ω_k^n de la siguiente manera:

$$f_i^{n+1} = \sum_k^N \omega_{k+1}^n S(x_i^{n+1} - X_k^{n+1})$$

Se regresa al paso 2 para el siguiente paso de tiempo. Estos pasos se repiten la cantidad de iteraciones que el estudio lo considere; en algunos artículos como en [10], donde se habla de la producción de un plasma a través de un láser se dice que la cantidad de iteraciones del proceso es de 100.000, pero ese tiempo se toma dependiendo del tipo de material que se piensa estudiar. En [2] este valor indica el tiempo de evaporación de un plasma de aluminio.

V. Implementación Paralela

El objetivo principal de este trabajo es reducir el tiempo de ejecución de la implementación secuencial del método semi-lagrangiano visto en la descripción anterior. Para cumplir este objetivo se usan unidades de procesamiento gráfico GPUs, y a su vez se realiza un análisis de los puntos paralelizables del algoritmo. En [11] se ve una implementación del método semilagrangiano utilizando CUDA sobre una tarjeta nVidia GTX 8800 lo que efectivamente demuestra que es posible realizar este tipo de implementaciones.

En el análisis se encontró que el calculo de la velocidad, posición, densidad de carga y potencial electroestático, se podía realizar de manera independiente para cada una de las partículas que se encuentran dentro del espacio de fase a estudiar, eso garantiza que varios hilos dentro de un dispositivo puedan realizar la soluciones de ecuaciones de Vlasov-Poisson de manera simultánea.

Por lo tanto teniendo en cuenta que el algoritmo tiene secciones altamente paralelizables, se procede a construir un programa en CUDA que permita realizar ésta tarea.

A. Preparación CUDA

Para este trabajo se decidió usar CUDA por su gran facilidad de uso con respecto a OpenCL; para programar con CUDA se siguen 3 pasos sencillos.

- Crear memoria local para las variables que se vayan a utilizar en el programa.
- Lanzar un código Kernel.
- Crear y liberar memoria

Para realizar un programa en CUDA se debe establecer la cantidad de hilos que se desean lanzar, posteriormente y dependiendo del tamaño del problema se deben crear bloques en los cuales se lanzarán el número de hilos establecido. La cantidad de bloques y el número de hilos que se pueden lanzar dependen fuertemente del tipo de GPU que se utilice [4], por este motivo estos valores podrían cambiar de acuerdo al hardware en el cual se implemente.

```
threadsPerBlock = 256;
blocksPerGrid =((NUMPARTICULAS) + threadsPerBlock - 1) / threadsPerBlock;
calculoVelocidad<<<blocksPerGrid, threadsPerBlock>>>(0.1,d_campoElectrico,d_v);
err = cudaGetLastError();
checkCudaError(err,"kernel calculoCampoElectrico");
```

Fig. 4: Hilos en CUDA.

1) Ejemplo: donde threadsPerBlock=256, en este caso se lanzarán 256 hilos por bloque: Es decir si se tienen 1000 partículas tendríamos lo siguiente

$$(1000) + 256 - 1/256 = 4,9$$

.

Lo que indica que se necesitan 4 bloques para ejecutar el kernel con la cantidad de Partículas propuestas.

B. Sistema de paralelizacion

Para la paralelización del algoritmo se logró identificar diferentes puntos:

$$\forall_k v_k^{n+1/2} = \frac{\triangle t}{2}(E(X_k^n, t^n)) + v_k^n Velocidad$$

$$\forall_k x_k^{n+1/2} = \triangle t v_k^{n+1/2} + v_k^n Posicion$$

Para cada partícula es posible calcular la nueva velocidad v y posición x de la siguiente forma en CUDA.

```
threadsPerBlock = 256;
blocksPerGrid =((NUMPARTICULAS) + threadsPerBlock - 1) / threadsPerBlock;
calculoVelocidad<<<blocksPerGrid, threadsPerBlock>>>(0.1,d_campoElectrico,d_v);
err = cudaGetLastError();
checkCudaError(err,"kernel calculoCampoElectrico");
```

Fig. 5: Cálculo velocidad y Posición.

Para los demás cálculos como la densidad de carga, potencial electrostático la paralelización funciona de la misma manera ya que cada calculo de las partículas lo hace cada hilo del procesador que se encuentra en cada bloque de la GPU, lo que hace que se este realizando varios procesos a la vez.

C. Cálculo de Rho

Antes de calcular el campo Eléctrico se debe realizar el calculo de la densidad de las partículas para luego obtener las nuevas posiciones y velocidades, el cálculo de rho se realiza de la siguiente manera como muestra la Figura 6.

En este punto se calcula los pesos para la densidad de carga de las partículas , para luego realizar el proceso que se nombra en el algoritmo secuencial.

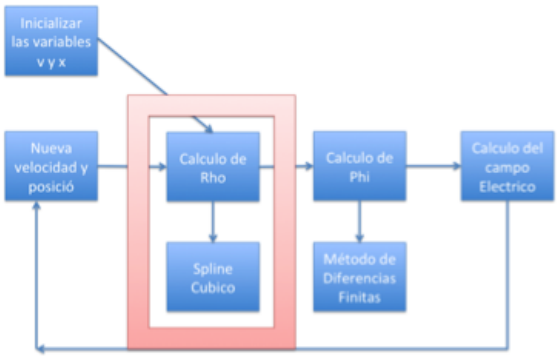

Fig. 6: Cálculo de la Densidad (rho)

```
status = culaDeviceSgetrs(c,numParticulasReal,nrhs,d_PHI,ldaPHI,d_ipivPHI,d_rhoPoisson,ldbPHI);
checkStatus(status);
```

Fig. 7: Cálculo con CULA.

Además la solución del sistema de ecuaciones resultante del método de diferencias finitas se realiza utilizando una librería llamada CULA de EM PHOTONICS, la cual utiliza llamados a CUDA. Al obtener el potencial electroestático para cada una de las partículas que se encuentran en el espacio de fase , se calcula el campo eléctrico para obtener nuevamente las posiciones y las nuevas velocidades hasta finalizar 100000 iteraciones.

```
calculoCampoElectrico<<<blocksPerGrid, threadsPerBlock>>>(0.1,d_rhoPoisson,d_campoElectrico);
err = cudaGetLastError();
checkCudaError(err,"kernel calculoCampoElectrico");
```

Fig. 8: Cálculo Campo Eléctrico.

Por lo tanto el proceso que se obtiene en el algoritmo paralelo es el mismo comparado con el secuencial solo que la diferencia está en que los cálculos de cada partícula son realizados por hilos diferentes sobre la GPU.

VI. Resultados

Los algoritmos fueron probados con diferentes números de iteraciones y variando el número de partículas a medida que se obtuvieron los datos para realizar las comparativas de aceleración. El proceso de pruebas se describirá a continuación. Las pruebas han seguido un protocolo, diseñado con el fin de llevar un orden en las distintas etapas del proceso.

A. Adquisición de los datos

Los algoritmos han sido probados sobre un equipo con las siguientes características como muestra la figura 9. sobre este equipo se ejecutaron los dos algoritmos cada uno de ellos aprovechando al máximo sus características, pero desde diferentes aspectos

Característica equipo de computo.
Procesador Intel core i7 3770K, 3,9 GHZ
16 GB de memoria RAM
nVidia GeForce GTX 580 Arquitectura Fermi
GPU 512 Núcleos,1536 MB-GD DR (1.5 GB memoria)

Fig. 9: Tabla de Caracteristicas.

B. Adquisición de tiempos y aceleración

En primer lugar se analizaron los resultados de los tiempos obtenidos y aceleración con respecto a la versión secuencial, para ello se muestra la tabla 10. con los tiempos de ejecución tanto del algoritmo secuencial como del algoritmo paralelo utilizando GPUs. Los valores que se obtienen son con respecto al número de iteraciones realizadas por el algoritmo.

Iteraciones	Partículas	CUDA	Secuencial	Aceleración
500	100	0m0.794s	0m0.161s	0,20277078
	200	0m1.425s	0m0.473s	0,33192983
	500	0m3.224s	0m2.605s	0,80800248
	1000	0m7.032s	0m10.288s	1,46302617
	3000	0m37.965s	1m34.284s	2,48344528
	7000	2m39.174s	8m32.186s	3,7203689
	10000	5m13.304s	17m20.777s	3,32193971
100000	100	1m57.239s	0m25.775s	0,21985005
	200	3m49.274s	1m39.488s	0,43392622
	500	9m32.575s	10m18.377s	1,07999301
	1000	22m39.242s	40m31.902s	1,78916043
	3000	124m53.331s	355m12.241s	2,84416116

Fig. 10: Comparativa Tiempo Secuencial y Paralelo.

A partir de los valores de la Tabla 10, se muestra la siguiente gráfica de aceleración con respecto al número de partículas con 500 iteraciones del algoritmo:

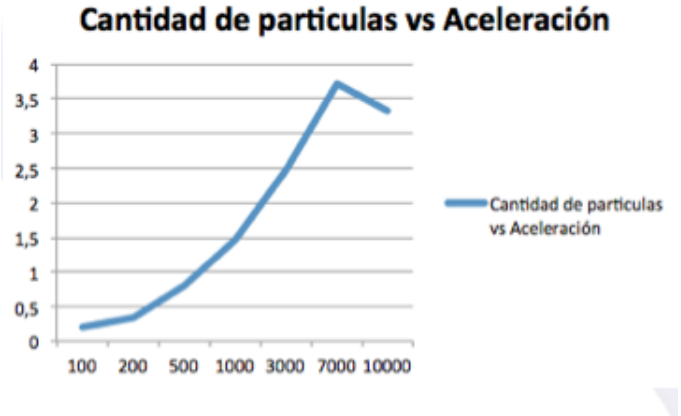

Fig. 11: Cantidad de Partículas vs Aceleración.

En la figura 11. Se logra analizar que a medida que se aumentaba el número de partículas la aceleración incrementa significativamente, también puede observarse que la aceleración en los intervalos menores a 100 partículas es menor

por que el cambio de contexto es mucho mas costoso pero aumenta consiguiendo una mejora de tiempo con respecto a la primera aproximación logrando acelerar el algoritmo con la GPU hasta 3.7x.

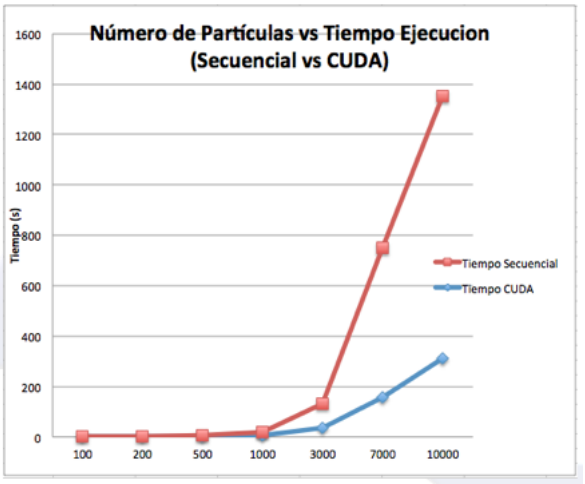

Fig. 12: Tiempo Secuencial y Paralelo.

De igual manera se hace un análisis de los tiempos de ejecución tanto de la CPU como la GPU. En la Figura 12 es notable que la CPU tiene un tiempo más elevando que la GPU a medida que aumentan los datos de entrada.

VII. Conclusiones

En el proceso de esta investigación se puede concluir que la aceleración de algoritmos puede ayudar a la obtención de resultados de una manera rápida y precisa, siempre y cuando el algoritmo a paralelizar no tenga un alto grado de dependencia de datos, para así poder mejorar su rendimiento en tiempo.

En este trabajo se logró una aceleración exitosa ya que se pudo tomar el algoritmo computacional semi-lagrangiano e implementarlo de forma secuencial para luego llevarlo a una solución paralela, por lo tanto se hicieron los siguientes pasos:

- Se desarrolló el algoritmo secuencial el cual se ejecutó en una CPU Intel Core i7 3770K y a la vez el algoritmo paralelo que se ejecutó en una tarjeta GTX 580 nVIDIA utilizando como marco de programación CUDA.
- Se utilizaron métodos de paralelización simples, en dónde lo que se hace es darle a cada núcleo del dispositivo en este caso la GPU una tarea distinta para realizar.
- Se logró una aceleración estable de 3.7 x, en donde el algoritmo paralelo mejora en cuanto al tiempo de ejecución frente al algoritmo secuencial.
- CUDA juega un papel fundamental en la paralelización de aplicaciones, ya que le facilita al programador poder rediseñar los algoritmos para el aprovechamiento de las nuevas arquitecturas computacionales
- Las características del método semi-lagrangiano presentan poca dependencia de datos, lo que permite que las GPUs puedan desarrollar de forma paralela la solución del sistema y el marco de programación CUDA permite tener acceso a este tipo de arquitecturas.
- Las GPUs cada vez están tomando mas importancia en el procesamiento de algoritmos para fines científicos, ya que permite obtener resultados en menor tiempo de ejecución como lo que se demostró en este proceso de investigación.

Aunque los resultados obtenidos son satisfactorios en el campo computacional, se puede concluir que este trabajo es un preámbulo al estudio de métodos para la solución de ecuaciones que estudian el movimiento de las partículas ya sea en un plasma o en un fluido, utilizando unidades de procesamiento gráfico para obtener los resultados en un menor tiempo de ejecución , pues a partir de esto, se pueden realizar muchas optimizaciones no solo para el método semi-lagrangiano si no también para PIC, N BODY, y el método de HydroDymamic Simulation.

Otra manera de optimizar el algoritmo paralelizado es en el uso de memoria local, por lo tanto es otro camino que queda abierto para el mejoramiento del rendimiento en cuanto a la forma de utilizar las unidades de procesamiento gráfico, esto queda planteado como un futuro trabajo para ver si se puede obtener una mejoría en el tiempo de ejecución del algoritmo.

Como trabajo futuro se puede realizar una comparativa, con respecto a los marcos de programaciónde OpenCL y CUDA, y evaluar el rendimiento computacional.

Agradecimientos

Queremos agradecer al Departamento Administrativo de Ciencia, Tecnología e Innovación Colciencias, a la Universidad Tecnológica de Pereira y a la convocatoria de Jóvenes Investigadores, por permitirnos explorar nuevas formas de conocimiento para aprovechar al máximo las nuevas tecnologías y así poder hacer sinergia con otros grupos de investigación dentro de la comunidad universitaria.

References

[1] A.Y Liu and M. Cohen , *Physics Rev.* . , 1990.
[2] T. Nedelea, J. Schou and H.M Urbassek *Plume expansion of laser-induced plasma studies with the particle-in-cell method*, Elsevier, 2002.
[3] T. Respaud, E. Sonnendrucker and N. Crouseilles *A Forward Semi-Lagrangian Method for Numerical Solution of the Vlasov Equation*, INRIA, 2008.
[4] nVidia *CUDA Programming Guide*, nVidia, 2009.
[5] H. Wang, E. Ewing *A Summary of Numerical Methods Time Dependent Advection Dominated Parcial Advection Dominated Partial Differential Equation*, Journal of Computational and Applied Mathematics, 2001.
[6] D.K. Purnell*Solution of Advective Equation Upstream Interpolation With a Cubic Spline*, Monthly Weather Review, 1976.
[7] N. Crouseilles, G. Latu *Hermite Spline Interpolation on Patches for a Parallel Solving of the Vlasov-Poisson Equation*, .
[8] M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi *A drift-kinetic semi- Lagrangian 4D code for ion turbulence simulation*, J. Comput. Phys, 2006.
[9] T. Respaud and E. Sonnedrucker *A Forward Semi-Lagrangian Method for the Numerical Solution of the Vlasov Equation*, INRIA, 2008.
[10] T. Nedelea, J. Schou *Plume Expansion of a Laser Plasma Studied with the PIC*, Applied Surface Science, 2002.
[11] V. Irakli, N. Evstigneev *Semi Lagrangian Method for Advection Equation on GPU in Unstructured R3 Mesh for Fluid Dynamics Application*, World Academy of Science, Engineering and Technology, 2009.