Asociación Argentina de Mecánica Computacional

Mecánica Computacional Vol XXVIII, págs. 273-286 (artículo completo)
Cristian García Bauza, Pablo Lotito, Lisandro Parente, Marcelo Vénere (Eds.)
Tandil, Argentina, 3-6 Noviembre 2009

MÉTODOS DE LATTICE BOLTZMANN SOBRE GPU PARA SIMULACIÓN DE FLUIDOS. COMPARACIÓN CON NS MEDIANTE ELEMENTOS FINITOS

Pablo R. Rinaldi[a,b], Enzo A. Dari[c], Diego D. Dalponte[a,b], Marcelo J. Vénere[b,c] y Alejandro, Clausse[a,b,c]

[a]CONICET, [b]UNCPBA, [c]CNEA

Palabras Clave: GPGPU, Lattice Boltzmann Methods, CUDA.

Resumen. En este trabajo se implementó el Método de Lattice Boltzmann (LBM) para la simulación del fluido en un canal con expansión súbita. Utilizando la tecnología Compute Unified Device Architecture (CUDA) de la empresa NVIDIA específica para resolver problemas computacionales complejos mediante el uso del hardware de gráficos. Los resultados se compararon con un modelo de elementos finitos "equal order" sobre una malla regular analizando diferentes salidas mostrando gran concordancia. El speed-up logrado con una placa GeForce 8800GT con respecto a una implementación de LBM equivalente en CPU es mayor al orden de magnitud.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

1 INTRODUCTION

El Método de Lattice Boltzmann (LBM) es un modelo de Fluidodinámica Computacional (CFD) que aproximan la solución de las ecuaciones de Navier-Stokes incompresibles (Chen y Doolen, 1998). En los últimos años, el LBM ha llamado considerablemente la atención como método de simulación eficiente para flujos complejos (Higuera y Succi, 1989; Karlin, Ferrante y Öttinger, 1999; Shan y He, 1998; Ansumali, Karlin y Öttinger, 2003).

La esencia de LBM, heredada de su predecesor, el autómata de Lattice Gas (Boon y Rivet, 2001), es simplemente un esquema de colisión y traspaso sobre las celdas de una grilla regular. Si bien el campo de aplicaciones de LBM ha crecido considerablemente, existen conceptos pendientes como la estabilidad, el refinamiento de la grilla y sobre todo la performance que dificultan la amplia aceptación de LBM para aplicaciones CFD.

Una característica fundamental de estos modelos es la posibilidad de realizar implementaciones en paralelo ya que al ser Autómatas Celulares (CA) con esquemas explícitos, un mismo código se ejecuta sobre todas las celdas del dominio para pasar de un estado del tiempo a otro.

Dentro de las tecnologías de computación en paralelo, una de las más novedosas es la utilización del hardware integrado en placas gráficas (GPU) para computación de alto desempeño. Las GPUs modernas están optimizadas para ejecutar una instrucción simple sobre cada elemento de un extenso conjunto en paralelo utilizando procesadores de shaders y técnicas de memoria compartida. Algunas de las últimas publicaciones en el área como (Goodnight, 2007; Tölke, 2007; Zhao, 2007) muestran la combinación GPU – LBM como válida para la simulación de fluidos

2 MÉTODO DE LATTICE BOLTZMANN

El concepto básico de los modelos LBM es construir un modelo cinético mesoscópico con variable interna discreta -i.e. velocidad- cuyas propiedades macroscópicas promediadas cumplan con las ecuaciones macroscópicas deseadas (Chen y Doolen, 1998).

2.1 Ecuación BGK

La ecuación de Lattice Boltzmann para el modelo denominado BGK (Bhatnagar, Gross y Krook, 1954), sobre una red cuadrada de dos dimensiones y nueve velocidades (d2Q9) se presenta como en las referencias (Chen, Chen, Martinez y Matthaeus, 1991; Qian, d'Humie`res y Lallemand, 1992):

$$f_i\left(x+\delta e_i, t+\delta\right)-f_i(x,t)=-\frac{1}{\tau}\left[f_i(x,t)-f_i^{(eq)}(x,t)\right], \qquad i=0,1,\ldots,8 \tag{1}$$

donde la ecuación está escrita en unidades físicas. Ambas escalas, espacial y temporal, tienen el valor $\boldsymbol{\delta}$ en unidades físicas. $f_i(x,t)$ es la función de distribución de densidad a lo largo de la dirección $\boldsymbol{e_i}$ en la celda ubicada en $\boldsymbol{x}$ en el tiempo ***t (x, t)***. La velocidad de las partículas $\boldsymbol{e_i}$ está dada por:

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

$$e_i = 0 \qquad i = 0$$

$$e_i = \left(\cos\left(\frac{\pi(i-1)}{2}\right),\ \sin\left(\frac{\pi(i-1)}{2}\right) \right) \qquad i = 1,2,3,4 \tag{2}$$

$$e_i = \sqrt{2}\left(\cos\left(\frac{\pi\left(i-4-\frac{1}{2}\right)}{2}\right),\ \sin\left(\frac{\pi\left(i-4-\frac{1}{2}\right)}{2}\right) \right) \qquad i = 5,6,7,8$$

La figura 1 muestra las direcciones de las velocidades en una celda del modelo d2Q9

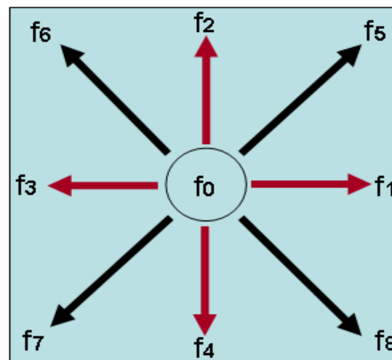

Fig. 1. LBM: Funciones de distribución en el modelo D2Q9.

2.2 Paso de Advección

La primera operación en cada iteración, en esta implementación particular, es el cálculo del avance de las partículas a las celdas vecinas a lo largo de sus direcciones de movimiento, es decir advección. En el paso de advección (Figura 2) cada sitio intercambia partículas con sus vecinos según la siguiente regla:

$$f^a(\vec{e},\vec{x},t) = f(\vec{e},\vec{x}-\vec{e}\,\delta t, t-\delta t) \tag{3}$$

donde f^a es la función de distribución. En este trabajo se utilizó un esquema “pull”, donde el paso de advección se realiza previo a la colisión y para cada celda se leen las ***fs*** entrantes de las celdas vecinas (Wellein, Zeiser, Hager y Donath; 2006).

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

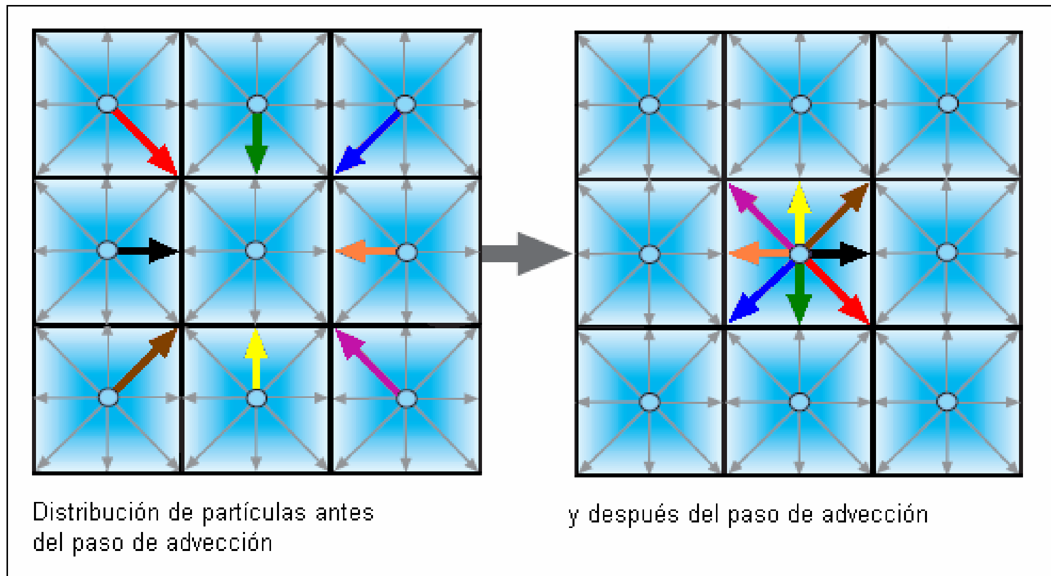

Figura 2. Paso de advección para la celda central

2.3 Paso de Colisión

El lado derecho de la ecuación 1 representa el término de colisión donde $\boldsymbol{\tau}$ es la relajación temporal simple que controla el rango de aproximación al equilibrio. La función de distribución de equilibrio $f_i^{(eq)}(x,t)$ depende solamente de la velocidad y densidad local y tienen la siguiente forma [15]

$$f_i^{(eq)} = t_i\rho\left[1+3(e_i\cdot u)+\frac{9}{2}(e_i\cdot u)^2-\frac{3}{2}u\cdot u\right],$$
$$t_0=\frac{4}{9},\ t_i=\frac{1}{9},\ i=1:4;\quad t_i=\frac{1}{36},\ i=5:8. \tag{4}$$

Donde la densidad de cada nodo, $\boldsymbol{\rho}$, y la velocidad macroscópica del fluido, $\boldsymbol{u=(u_x, u_y)}$, se definen en términos de la función de distribución de la siguiente manera:

$$\sum_{i=0}^{8} f_i=\rho,\qquad \sum_{i=0}^{8} f_i e_i=\rho u \tag{5}$$

La presión está dada por $p=c_s^2\rho$, donde $\boldsymbol{c_s}$ es la velocidad del sonido con $c_s^2=\frac{1}{3}$, y la viscosidad cinemática v está dada por:

$$\nu=\left[\frac{(2\tau-1)}{6}\right]\delta \tag{6}$$

2.4 Condiciones de contorno

Una de las cuestiones más complejas de la implementación de LBM son las condiciones de contorno. En las paredes se utilizó una condición de no deslizamiento, lo que significa velocidad neta cero en todas las direcciones. Para la implementación se

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

siguió el esquema básico on-grid bounce-back (Succi, 2001) (figura 3). Es sabido que directamente revertir las ***fs*** hacia el fluido da una precisión de primer orden, por eso se implementó una versión modificada propuesta en (Zou y He, 1997). Por ejemplo, las ***fs*** desconocidas para un nodo en la pared superior se definen de la siguiente forma:

$$f_4 = f_2, \;\; f_7 = f_5 + \frac{1}{2}(f_1 - f_3), \; f_8 = f_6 - \frac{1}{2}(f_1 - f_3) \tag{7}$$

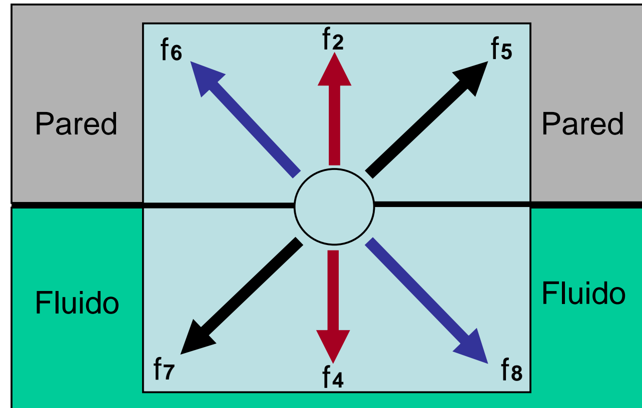

Figura 3. Condiciones de contorno on-grid bounce back

De manera similar a la ecuación 7 se calcula en las demás paredes, partiendo de que para que la velocidad sea cero se debe cumplir:

$$\rho u_x = f_1 + f_5 + f_8 - f_3 - f_6 - f_7 = 0, \quad \rho u_y = f_2 + f_5 + f_6 - f_4 - f_7 - f_8 = 0 \tag{8}$$

Para forzar el flujo a través del canal se utilizó el esquema de velocidad presión utilizado en (Zou y He, 1997) con una leve modificación ya que se forzó un perfil parabólico cuadrático en la entrada. Este esquema se conoce como la condición Velocity-Pressure Boundary (VPB), en el cual se define la velocidad a la entrada del dominio y la presión a la salida. Sobre la entrada la componente de velocidad transversal al flujo es nula y solamente se fija la componente axial ux de la velocidad. Para una entrada en la dirección ***x*** se tiene entonces:

$$\rho = \frac{f_0 + f_3 + f_7 + 2(f_5 + f_4 + f_6)}{1 - u_x} \tag{9}$$

$$f_1 = f_5 + \frac{2}{3}\rho u_x \tag{10}$$

$$f_2 = f_6 + \frac{1}{6}\rho u_x - \frac{1}{2}(f_3 - f_7) \tag{11}$$

$$f_8 = f_4 + \frac{1}{6}\rho u_x + \frac{1}{2}(f_3 - f_7) \tag{12}$$

A la salida se fija la presión, que como es proporcional a la densidad equivale a fijar un $\boldsymbol{\rho}$ constante. Además, se asume que la componente en la dirección y de la velocidad es 0. Entonces, para una salida en la dirección positiva de las ***x*** queda:

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

$$u_x = \frac{f_0 + f_3 + f_7 + 2(f_1 + f_2 + f_8)}{\rho} - 1 \tag{13}$$

$$f_5 = f_1 - \frac{2}{3}\rho u_x \tag{14}$$

$$f_4 = f_8 - \frac{1}{6}\rho u_x - \frac{1}{2}(f_3 - f_7) \tag{15}$$

$$f_6 = f_2 - \frac{1}{6}\rho u_x + \frac{1}{2}(f_3 - f_7) \tag{16}$$

3 EXPANSION SUBITA

Partiendo del modelo de presentado, se simuló el flujo en un canal de dos dimensiones con una expansión súbita como lo muestra la figura 4.

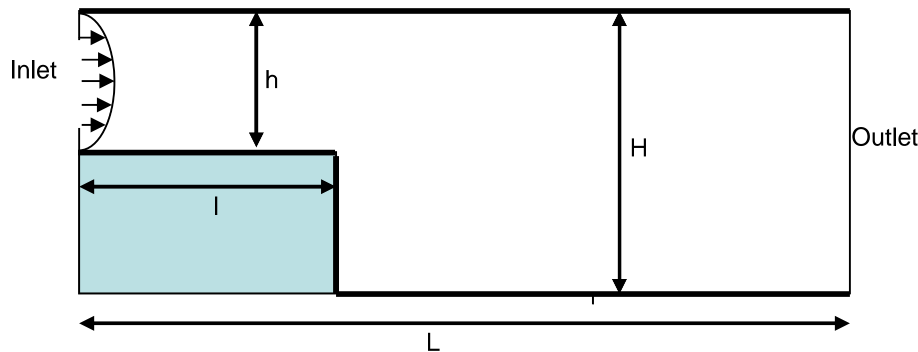

Figura 4. Canal con expansión súbita en el primer tercio

La geometría elegida fue la siguiente: La longitud total del canal (L) es de 3, el ancho del canal es de 0,5 en la entrada (h) y de 1 en la salida (H), con lo cual la expansión ubicada a distancia 1 de la entrada (l) es de 0,5. El flujo de entrada es un perfil parabólico con una velocidad adimensional máxima central $\boldsymbol{u_x}$ de 0,3 y las condiciones en la salida son de presión constante y velocidad vertical $\boldsymbol{u_y}$ = 0. Los valores adimensionales del fluido simulado son de una densidad $\boldsymbol{\rho}$ = 1,0 y una viscosidad $\boldsymbol{\mu}$ = 0,003 con lo cual la viscosidad cinemática se calcula es $\boldsymbol{\nu}$ = 0,003. Entonces, si suponemos una profundidad (***z***) muy grande con respecto al alto del canal de entrada, el diámetro hidráulico de la entrada se calcula como (White, 1988):

$$dh = \frac{4A}{P} = \frac{4hz}{2h + 2z} \cong \frac{4hz}{2z} = 2h \tag{17}$$

y el número de Reynolds para ese se diámetro hidráulico es:

$$\mathrm{Re}_{dh} = \frac{U_{\max} dh}{\nu} = 100 \tag{18}$$

4 LA UNIDAD DE PROCESAMIENTO GRÁFICO O GPU

La GPU es el chip de las placas gráficas que efectúa las operaciones requeridas para renderizar píxeles en la pantalla y están optimizadas para ejecutar una instrucción simple sobre cada elemento de un extenso conjunto simultáneamente (*i.e.* Instrucción

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

Simple y Múltiples Datos (SIMD)). En los últimos años la performance de las GPUs ha excedido la ley de Moore por más de 2.4 veces al año (Moreland y Angel, 2003).

4.1 GPU NVIDIA GeForce serie 8

La serie GeForce 8 de placas gráficas de la empresa NVIDIA fue desarrollada en conjunto con el modelo de programación CUDA (NVIDIA 2008) de la misma marca. Las GPUs de esta línea están compuestas por un conjunto de multiprocesadores con arquitectura SIMD, como se ilustra en la figura 5. Esto significa que cada uno de los procesadores dentro de los multiprocesadores ejecuta la misma instrucción en cada ciclo de reloj, pero sobre diferentes datos simultáneamente.

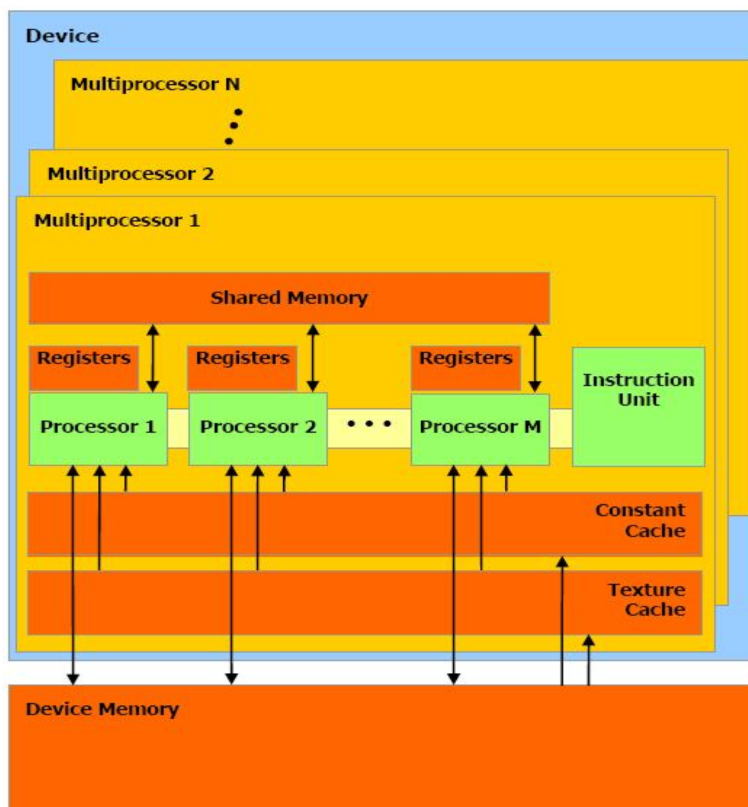

Figura 5: Conjunto de multiprocesadores SIMD con memoria compartida en una GPU NVIDIA Geforce 8000 (NVIDIA, 2008a).

Las características de la GPU NVIDIA GeForce 8800 GT utilizada en este trabajo se listan en la tabla 1.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

NVIDIA Geforce 8800 GT	
Multiprocesadores	14
Procesadores de Streams	112
Tamaño de la Memoria	512 MB
Interfaz de Memoria	256 bits
Ancho de Banda de Memoria	57,6 GB/s
Máxima Performance Aproximada	200 Gflops
Velocidad del núcleo	600 Mhz
Velocidad de los shaders	1500 Mhz
Velocidad de la memoria	900 Mhz
Compatibilidad versión CUDA	1.1

Tabla 1: Características de la GPU utilizada (NVIDIA, 2008a).

4.2 Modelo de programación CUDA

La tecnología CUDA de NVIDA es una nueva arquitectura que permite a la placa gráfica resolver problemas computacionales complejos. CUDA facilita el acceso de aplicaciones con alto costo computacional al poder de procesamiento de las GPU NVIDIA a través de una nueva interfaz de programación. La utilización del lenguaje C estándar simplifica en gran medida el desarrollo de software. La tecnología CUDA es compatible con los sistemas operativos Linux y Windows® XP®. En este trabajo se utilizó la versión 1.1 sobre plataforma Debian GNU/Linux OS.

La GPU, denominada *device* se ve como un dispositivo computacional capaz de ejecutar un gran número de hilos de ejecución en paralelo (*threads*). Opera como un coprocesador para la CPU principal denominada *host*. Las porciones de aplicaciones de cálculo intensivo corriendo en la CPU principal se descargan al dispositivo utilizando una función que se ejecuta en la placa en forma de *threads* múltiples paralelos. Tanto el *host* como la GPU mantienen su propia memoria RAM, llamadas *host memory* y *device memory*, respectivamente. Se pueden copiar datos de una RAM a la otra a través de métodos optimizados que utilizan el acceso directo a memoria (DMA) de alta performance de la propia placa.

4.3 Implementación LBM

El programa consta de una aplicación principal desarrollada en C que inicializa los datos, crea las grillas y realiza las invocaciones de las dos versiones del algoritmo LBM. Una versión de ejecución paralela en CUDA que se denomina *kernel* y otra secuencial en C utilizada para comparaciones de tiempos de ejecución.

Se utilizaron 18 arreglos de punto flotante (uno para cada $\boldsymbol{f}$ en $\boldsymbol{t}$ y en $\boldsymbol{t+1}$) que se crean dentro de un programa C y luego se copian a memoria del dispositivo. Con estos arreglos se representa una grilla rectangular de 32 x 96 celdas, ya que por cuestiones de performance es recomendable utilizar múltiplos de 16 en las estructuras principales (NVIDIA, 2008b). Al utilizar *on-grid bounce-back* en las paredes del canal, éstas quedan ubicadas en el centro de la celda, con lo cual un canal simulado entre dos paredes tiene una celda menos que las utilizadas para su representación. Por lo tanto la relación de escala final máxima posible para las dimensiones elegidas es $\boldsymbol{\delta} = 1/30$ lo que corresponde a un canal de entrada de 15 celdas. Si tomáramos un canal de entrada de 16 celdas (relación 1/32), serían necesarias 33 celdas para representar el canal de salida.

Para definir la geometría, se utiliza un arreglo de enteros, de las mismas dimensiones que las grillas de $\boldsymbol{fs}$ donde se especifica el tipo de celda, esto permite cambiar

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

geometrías sin variar el resto del código. Para este problema en particular se utilizaron 12 tipos de celdas diferentes:

1. Flujo: Celdas internas al modelo, advección normal.
2. Inlet: Celdas de la pared izquierda donde se aplica la condición de flujo de entrada
3. Outlet: Celdas de la pared derecha donde se aplica las condiciones de salida.
4. Borde Superior: Bounce back hacia abajo con velocidad cero.
5. Borde inferior: Bounce back hacia arriba con velocidad cero.
6. Borde izquierdo: Bounce back hacia la derecha con velocidad cero.
7. Borde superior izquierdo del canal: condiciones de inlet y bounce back.
8. Borde inferior izquierdo del canal: condiciones de inlet y bounce back.
9. Borde superior derecho: condiciones de outlet y bounce back.
10. Borde inferior derecho: condiciones de outlet y bounce back.
11. Vértice de la expansión: Bounce back hacia arriba y hacia la derecha.
12. No se opera, celdas fuera del dominio.

4.4 Algoritmo

El método de Latticce Boltzmann presentado en el punto 2 consiste en dos loops anidados a lo largo de las dos dimensiones de la grilla que calcula para cada celda la advección, las condiciones de contorno y por último el paso de colisión y relajación. Al realizar todos los pasos del algoritmo como uno solo, se reduce notablemente la cantidad de datos transferidos desde y hacia la memoria principal. El uso de dos copias de los datos una para ***t*** y otra para ***t+1*** facilita la paralelización eliminando la dependencia de datos. Los datos se leen de una copia y se escriben en otra durante un paso del algoritmo completo.

Los *threads* de CUDA se agrupan en bloques o (*thread blocks*) y pueden cooperar entre sí compartiendo eficientemente datos a través de la memoria compartida de acceso rápido y sincronizando sus ejecuciones. Para explotar el potencial del hardware eficientemente un bloque debe contener por lo menos 64 *threads* y no más de 512 (Tölke, 2007). Los bloques que ejecuten el mismo código o *kernel* pueden ser agrupados en una grilla de bloques, con lo que el número de *threads* que pueden ser lanzados en un solo llamado es mucho mayor. Pero los *threads* de diferentes bloques no pueden comunicarse de forma segura o sincronizarse entre sí. Los diferentes bloques de una grilla pueden correr en paralelo y para aprovechar el hardware eficientemente se deberían utilizar al menos 16 bloques por grilla (Tölke, 2007). La sincronización de *threads* de un mismo bloque se hace mediante la función *syncthreads()* que define un punto de sincronización. Una vez que todos los *threads* alcanzan este punto, la ejecución continúa normalmente.

4.5 Acceso a Memoria

Los *threads* de CUDA pueden acceder datos de diferentes espacios de memoria durante su ejecución como lo muestra la figura 5. Cada *thread* tiene su memoria local privada, cada bloque de *threads* tiene su memoria compartida o *shared* visible a todos los *threads* del bloque y finalmente, todos los *threads* pueden acceder a la misma memoria gloabal o *device memory*. Lleva 4 ciclos de reloj acceder a un dato en memoria compartida, mientras que la latencia de leer un dato de memoria global es de entre 400 y 600 ciclos. Pero el *scheduller* de *threads* puede esconder gran parte de esta demora si hay suficientes instrucciones aritméticas independientes que puedan realizarse durante ese lapso. Por eso es recomendable lanzar varios bloques de *threds* por cada

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

multiprocesador para que cuando un bloque queda en espera de datos, otro tome su lugar en el multiprocesador.

Por otro lado, el ancho de banda de cada espacio de memoria depende significativamente del patrón de acceso. Como la memoria global es mucho más lenta y no tiene caché, el acceso debe ser alineado y minimizado copiando datos a memoria local o compartida para su posterior procesamiento. Si *threads* de un mismo bloque acceden simultáneamente a memoria en direcciones alineadas estos accesos se agrupan en uno solo por cada 16 *threads* aumentando en gran medida el ancho de banda. Para el código LBM implementado en este trabajo, la actualización de una celda se compone de los siguientes pasos:

1. Leer las funciones de distribución de las celdas adyacentes en memoria global, i.e. $f_i(\vec{x}-\vec{e}\delta t, t-\delta t)$ a memoria compartida. Esto se hace de manera alineada salvo en los límites de la grilla.
2. Sincronizar.
3. Aplicar las condiciones de contorno para obtener las *fs* faltantes.
4. Calcular ρ, $\vec{u}$ y $f_i^{(eq)}$
5. Sincronizar.
6. Escribir los valores actualizados en la celda actual, i.e. $f_i(\vec{x}, t)$ a memoria global. Estos accesos son alineados en todos los casos.

Como la grilla utilizada es de 32 x 96 celdas, la forma más eficiente de utilización de la placa es ejecutar todas las celdas en paralelo maximizando el nivel de paralelización. De este modo, cada *thread* se encarga de la actualización de una sola celda y se generan 32 bloques de 64 o 128 *threads* cada uno.

5 RESULTADOS

5.1 Elementos Finitos

Para validar los resultados se realizó una simulación con un programa que resuelve las ecuaciones de Navier-Stokes con un modelo de elementos finitos *equal-order* igual interpolación para velocidad y presión, con estabilización sub-grid scale (SGS). Este programa está linealizado por atraso del término convectivo, por lo tanto resuelve un estacionario mediante un pseudo-transitorio. Se utilizó una malla regular de 1821 nodos, los cuales forman elementos triangulares. Las condiciones de contorno se fijan sobre 137 elementos. Se simularon 100 segundos en ambos modelos (FE y LBM) partiendo del estacionario. Sólo se ejecutaron 100 pasos del modelo FE mientras que LBM requirió 3000 iteraciones.

5.2 Campos de velocidades

A continuación se muestran diferentes visualizaciones del campo de velocidades para cada uno de los modelos luego de 100 segundos de la simulación. Estas vistas representan un análisis cualitativo ya que los valores absolutos son diferentes y también lo es la relación de éstos con la escala de colores.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

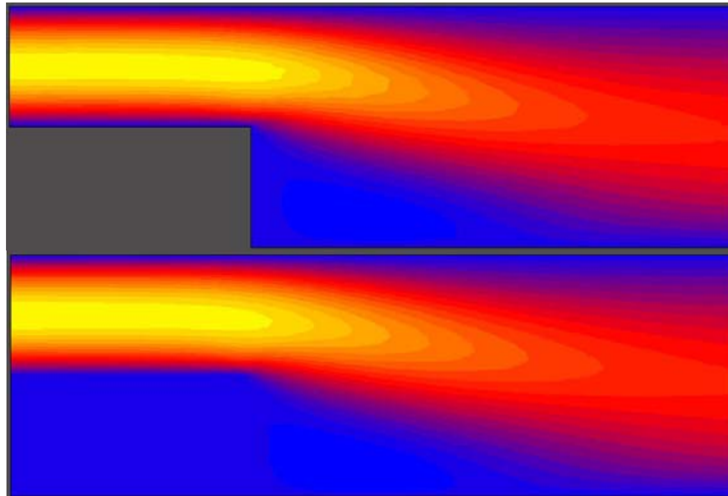

Figura 6: Comparación de la velocidad en x ($\boldsymbol{u_x}$) por gráfico de zonas de velocidad. La versión de Elementos Finitos es la imagen superior, la de Lattice Boltzmann en GPU la inferior

En la primera imagen (figura 6) se aprecia claramente la concordancia de ambos modelos en cuanto a los perfiles de velocidades $\boldsymbol{u_x}$. La velocidad máxima del centro se reduce al pasar la expansión súbita (va de amarillo a rojo) y se dirige hacia el centro del canal.

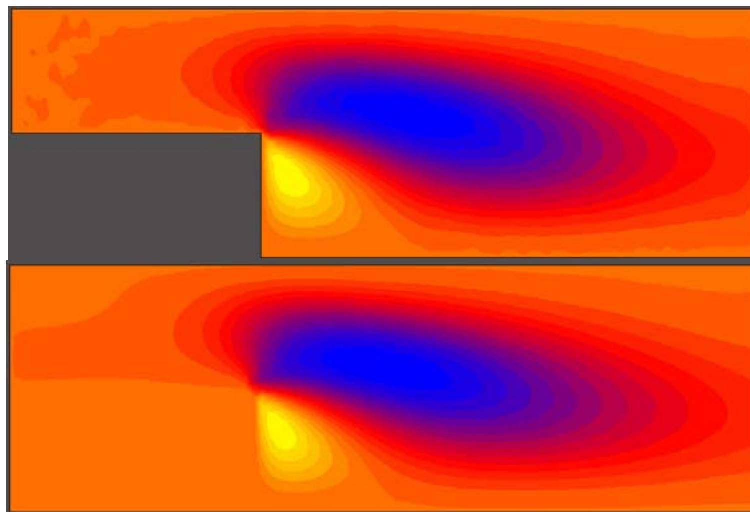

Figura 7: Comparación de la velocidad en y ($\boldsymbol{u_y}$) por gráfico de zonas de velocidad. La versión de Elementos Finitos es la imagen superior, la de Lattice Boltzmann en GPU la inferior

En la figura 7 pude verse como la velocidad en $\boldsymbol{y}$ ($\boldsymbol{u_y}$) es prácticamente nula en la entrada del canal y sobre la salida en ambos modelos (color naranja). El flujo alcanza un máximo positivo después de pasar por la expansión correspondiente al vórtice (amarillo) y un máximo negativo en el centro del canal (color azul). Se observa concordancia entre ambos modelos.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

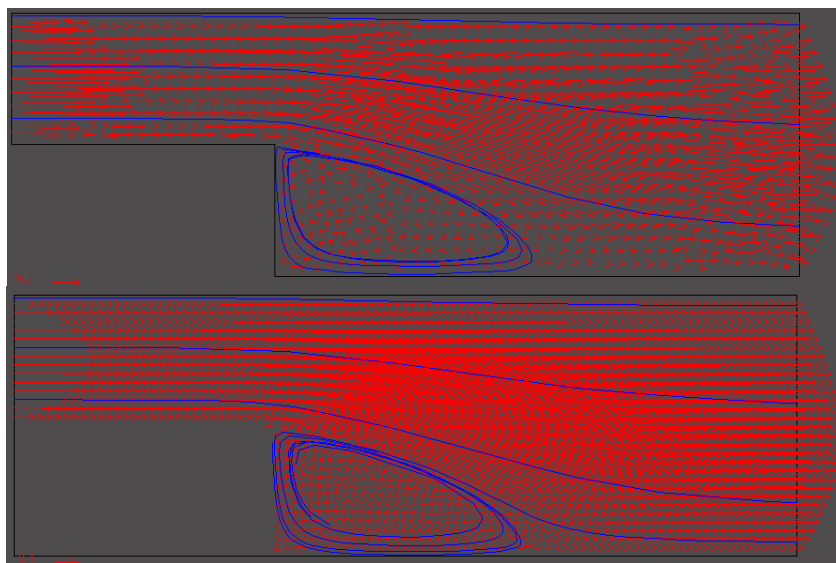

Figura 8: Comparación de los campos de velocidades y líneas de corriente. FE arriba, LBM-GPU debajo.

En esta última comparación (figura 8) se muestra el campo de velocidades en ambos casos. Las líneas de corriente (azules) describen patrones similares y en ambos casos se observa el vórtice que se genera en la esquina inferior del la expansión.

5.3 Perfiles de velocidades

Para evaluar cuantitativamente los resultados, se midieron los perfiles de velocidad en diferentes posiciones a lo largo del canal. A continuación se presentan los gráficos correspondientes a estos perfiles de velocidad ux normalizados superpuestos.

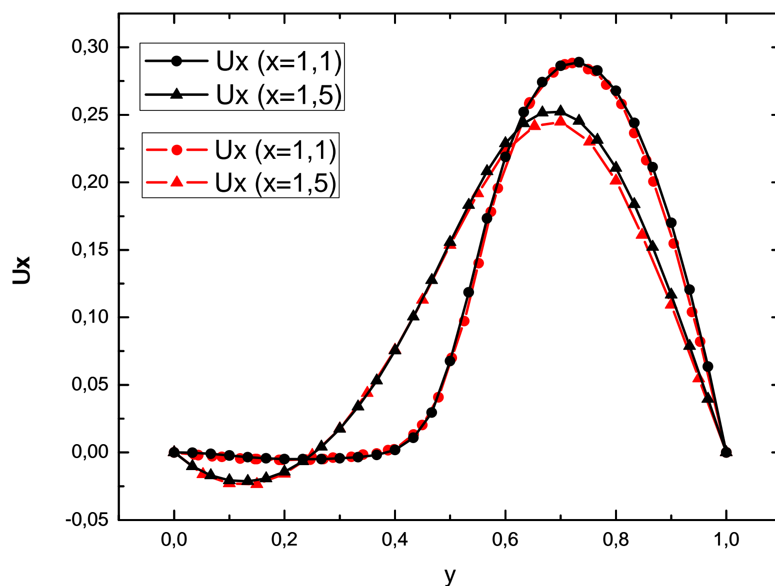

Figura 9: Velocidad $\boldsymbol{u_x}$ normalizada a lo largo de un corte vertical del canal inmediatamente después de la expansión (x = 1,1) y en el centro del canal (x = 1,5). Comparación entre ambas simulaciones. LBM en negro, FE en rojo.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

En la figura 9 se observa el perfil de velocidad inmediatamente después de la expansión (donde conserva el perfil impuesto) y en el centro del canal donde la velocidad se vuelve levemente negativa entre y=0 e y=0,2. Este retroceso del fluido produce el vórtice señalado anteriormente.

5.4 Tiempos de ejecución:

La unidad más utilizada en la medición de performance de modelos LBM es el número de celdas actualizadas en un segundo o LUPS (Lattice-Site Updates per Second) (Lammers y Küster, 2007). En la tabla 2 se muestra la comparación de tiempo promedio de una iteración para toda la grilla y los correspondientes MLUPS aproximados para las versiones del modelo LBM en C y CUDA ejecutados sobre las correspondientes plataformas (CPU y GPU) y el speedup alcanzado.

Tamaño de la Grilla	Bloques	Tiempo de CPU (ms)	MLUPS CPU	Tiempo de GPU (ms)	MLUPS GPU	Speedup
(96x32) 3072	32	0,59	5	0,041	75	14,4

Tabla 2. Tiempos de ejecución para el modelo de Lattice Boltzmann.

El modelo LBM en CUDA y el de elementos finitos fueron también ejecutados utilizando el comando *time* para calcular el tiempo total insumido por la aplicación en una corrida. Lo que se pretende mostrar es que el tiempo total que nos insume calcular una solución a un problema determinado utilizando esta nueva tecnología es viable desde el punto de vista del usuario final y que está dentro de los márgenes actuales de una simulación CFD. El código LBM presentado fue desarrollado y optimizado para la simulación en 2D mientras que el solver de FE utilizado es una compleja herramienta de simulación con áreas de aplicación mucho más amplias.

time	FE – CPU	LBM – CPU	LBM – GPU
Real	42,39s	1,77s	0,123s

Tabla 2. Tiempos totales de las simulaciones en los diferentes modelos.

Estos tiempos de ejecución corresponden a máquinas pc de escritorio con procesador Intel Core2 Quad de 2.4 Ghz y 8 gb. de memoria Ram.

6 CONCLUSIONES

En este trabajo se planteó una estrategia alternativa a los modelos clásicos de Fluidodinámica Computacional utilizando un método de Lattice Boltzman implementado en paralelo sobre hardware de gráficos. Se validó el modelo con la simulación de un problema clásico de mecánica de fluidos como lo es el flujo en un canal con expansión súbita. Los resultados muestran que el enfoque es válido en cuanto a la precisión de los cálculos realizados y que los tiempos de ejecución son correctos en comparación con un modelo que resuelve Navier-Stokes por Elementos Finitos. El speedup conseguido en comparación con una implementación equivalente para CPU es de más de catorce veces. Si bien se trata de un ejemplo simple, los resultados obtenidos son muy alentadores en cuanto a que la implementación de LBM sobre placas gráficas es una poderosa herramienta para la simulación de fluidos.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

7 REFERENCIAS

Ansumali S., Karlin I. V. and Öttinger H. C. Consistent Lattice Boltzmann Method, *Europhys. Lett*. 63, 798, 2003.

Bhatnagar, P., Gross, E. and Krook, M. A model for collisional processes in gases I: small amplitude processes in charged and neutral one-component system. *Phys. Rev.* 94, 511, 1954

Boon J.-P. and Rivet J.-P., Lattice Gas Hydrodynamics. *Cambridge University Press,* Cambridge, 2001.

Chen, S., Chen, H., Martinez, D. O. and Matthaeus, W. H. Lattice Boltzmann model for simulation of magnetohydrodynamics. *Phys. Rev. Lett.* 67, 3776, 1991

Chen S. and Doolen G. D. Lattice Boltzmann Methods for Fluid Flous. *Annu. Rev. Fluid Mech.* 30, 329, 1998

Goodnight, N. CUDA/OpenGL Fluid Simulation. (Online): http://new.math.uiuc.edu/MA198-2008/schaber2/fluidsGL.pdf 2007.

Higuera F. and Succi S., Simulating the flow around a circular cylinder with a lattice Boltzmann equation, *Europhys, Lett.* 8, 517, 1989

Karlin I. V., Ferrante A. and Öttinger H. C. Perfect entropy functions of the Lattice Boltzmann method, *Europhys. Lett.* 47, 182, 1999.

Lammers, P. and Küster U. Recent Performance Results of the Lattice Boltzmann Method, *High Performance Computing on Vector Systems* 2006. Part 2. 51-59. 2007.

Moreland, K. and Angel, E. The FFT on a GPU. SIGGRAPH/EUROGRAPHICS *Conference On Graphics Hardware*, M. Doggett, W. Heidrich, W. Mark, and A. Schilling, Eds., Eurographics Association. San Diego, California. Eurographics Association. pp. 112–119, 2003.

NVIDIA, NVIDIA CUDA Home Page. (Online): http://www.nvidia.com/object/cuda_home.html 2008a.

NVIDIA. NVIDIA CUDA Compute Unified Device Architecture – Programing Guide. (Online): http://developer.download.nvidia.com 2008b.

Qian, Y., d'Humie`res, D., and Lallemand, P. Recovery of Navier–Stokes equations using a lattice-gas Boltzmann method. *Europhys. Lett.* 17, 479, 1992.

Shan X. and He X., Discretization of the Velocity Space in the Solution of the Boltzmann Equation. *Phys. Rev. Lett.* 80, 65, 1998.

Succi S. The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Numerical Mathematics and Scientific Computation. *Clarendon Press*, Oxford, 2001.

Tölke, J. Implementation of a Lattice Boltzmann kernel using the Compute Unified Architecture Developer by nVIDIA. (Online): http://www.irmb.tu-bs.de/UPLOADS/toelke/Publication/toelked2q9.pdf 2007.

Wellein, G., Zeiser, T., Hager, G., Donath, S. On the single processor performance of simple lattice Boltzmann kernels, *Computers & Fluids*, vol. 35, 910-919 (2006)

White F. M. Mecánica de Fluidos. McGraw-Hill eds. ISBN 968-451-581-2. 1988.

Zhao, Ye. Lattice Boltzmann based PDE Solver on the GPU. *Visual Comput 2007*. Springer-Verlag, 10.1007/s00371-007-0191-y, 2007.

Zou, Q. and He, X., On pressure and velocity boundary conditions for the lattice Boltzmann BGK model, *Phys. Fluids*, vol.9 (6), pp.1591-1598, 1997.

Copyright © 2009 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar