Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXX, págs. 349-360 (artículo completo)
Oscar Möller, Javier W. Signorelli, Mario A. Storti (Eds.)
Rosario, Argentina, 1-4 Noviembre 2011

CALB: UNA CAPA DE ABSTRACCIÓN PARA MOTORES DE LATTICE-BOLTZMANN

Javier Dottori, Gustavo Boroni, Diego Dalponte y Alejandro Clausse

Universidad Nacional del Centro de la Provincia de Buenos Aires y CNEA-CONICET-CICPBA
jdottori, gboroni, ddalpont, clausse@exa.unicen.edu.ar

Palabras clave: Motores de Lattice Boltzmann, Capa de abstracción, Simulación de fluidos.

Resumen. En este trabajo se presenta el diseño de una Capa de Abstracción de *Lattice* Boltzmann (CALB) para utilizar diferentes motores de *Lattice* Boltzmann (LB). CALB permite generalizar los modelos en cualquier número de dimensiones, facilitando así el desarrollo de nuevas versiones, o la elección de alguna preexistente para la implementación de una aplicación concreta. El objetivo es minimizar el esfuerzo de desarrollo de nuevos motores fomentando la reutilización de código y el testeo, permitiendo de este modo configurar experimentos existentes en nuevos motores, y comparar la performance entre los mismos. El diseño de CALB conserva la facilidad de uso de lenguajes orientados a objetos y propone una interfaz uniforme respecto de otros motores desarrollados. Como resultado, CALB permite utilizar implementaciones en lenguajes de bajo nivel como FORTRAN, o que se ejecutan en placa gráfica (GPU). Por último, se presenta un estudio de la penalización introducida al utilizar la CALB.

1 INTRODUCCIÓN

En este trabajo se utiliza el Método de Lattice Boltzmann (LBM) para la simulación de fluidos. El LBM es un autómata celular basado en una grilla regular de celdas (Chen y Doolen, 1998), cuyo estado está representado poblaciones de partículas clasificadas dentro de un conjunto finito de velocidades. El algoritmo itera un conjunto de reglas microscópicas que representan el transporte e interacción entre partículas, y a partir de dichas reglas se simula el comportamiento macroscópico del sistema. LBM es utilizado para resolver una gran variedad de ecuaciones diferenciales parciales para modelar diferentes fenómenos físicos.

La evolución constante en la capacidad de procesamiento de las computadoras, ha permitido recientemente el uso del LBM en simulaciones interactivas de tiempo real (García Bauza, 2010). Por esta razón es que ha comenzado a aplicarse en la industria del entretenimiento, videojuegos y simuladores.

El LBM puede ser planteado desde diferentes enfoques, siendo su desarrollo muy amplio, interdisciplinario y variado. La escasa distancia entre la teoría física representada y su implementación hacen que el ciclo de vida del desarrollo sea muy corto, lo cual le da una actividad de investigación muy activa (LBMethod.org).

Dentro de la ingeniería de software se han desarrollado diferentes motores de LBM. En cada uno de ellos si bien se tienen en cuenta la performance numérica y calidad del software se persiguen diferentes objetivos. Por ejemplo, hoy en día se puede encontrar una gran cantidad de implementaciones ejecutando sobre placas gráficas (Rinaldi, 2010). Dadas las distintas líneas de investigación que trabajan con LBM y las características modeladas en cada situación resulta entendible que el número de motores existentes tenga un crecimiento muy grande, y por lo tanto elegir entre uno y otro no es un trabajo fácil. Por otro lado, dado que muchas veces se pueden agregar las características necesarias a un motor ya existente, tampoco se pueden descartar los motores por el simple hecho de no soportar una funcionalidad deseada. Por otro lado, cambiar de motor durante la etapa de desarrollo puede resultar una tarea difícil, ya que generalmente ofrecen interfaces muy diferentes.

En este trabajo se propone una capa de abstracción para diseñar, implementar y/o reutilizar motores de lattice Boltzmann (CALB). Dicha capa de abstracción facilita el desarrollo de nuevas versiones, o la elección de alguna preexistente para una aplicación concreta. Los objetivos se centran en minimizar el esfuerzo de cambio y fomentar la reutilización de código.

Para el diseño, implementación y ensayo de CALB se utilizaron diferentes motores de LBM disponibles, implementados en diferentes lenguajes de programación, incluyendo uno implementado en CUDA (Rinaldi, 2011).

Este trabajo está organizado de la siguiente manera: en la sección 2 se describen las principales características del método, en la sección 3 se explica el diseño realizado, en la sección 4 se detallan los experimentos realizados y por último se muestran las conclusiones y los posibles trabajos futuros.

2 MÉTODO DE LATTICE BOLTZMANN

LBM se basa la ecuación discreta de Boltzmann que modela el transporte de partículas a nivel mesoscópico, de manera que las propiedades macroscópicas promediadas obedezcan las ecuaciones macroscópicas deseadas (Kadanoff, 1986).

En cada celda espacial y cada paso tiempo el estado físico se representa con un conjunto finito de números reales asociados con la distribución de partículas. Si bien el método utiliza más memoria que los métodos tradicionales de resolución numérica, brinda numerosas ventajas, entre las que se pueden destacar la facilidad en la creación de algoritmos que se adaptan naturalmente a varias arquitecturas de software y hardware, así como una eficiente

paralelización de las simulaciones debido al carácter explícito del esquema. Otras ventajas adicionales son la capacidad de representar fácilmente fenómenos físicos complejos**,** que van desde flujos de dos fases hasta interacciones químicas. Como el método tiene sus orígenes en la descripción molecular de un fluido se puede incorporar directamente términos físicos derivados de una interacción ya conocida entre moléculas. Por esta razón, LBM se ha vuelto muy popular en la investigación, ya que mantiene corto el ciclo entre la elaboración de la teoría y la formulación del modelo numérico correspondiente (LBMethod.org).

La variable de estado de un esquema LBM es la población de partículas $f_i(\vec{x},t)$, donde $\vec{x}$ es la posición de la celda, t en el tiempo e i un índice que indica la velocidad en un conjunto finito $\{e_i|0 \le i < q\}$. La evolución de f_i está gobernada por la siguiente regla:

$$f_i\left(\vec{x} + \mathrm{e_i}\overrightarrow{\Delta \mathrm{x}}, t + \Delta \mathrm{t}\right) = f_i(\vec{x},t) + \Omega_\mathrm{i}, \quad i = 0,1,\dots,q-1 \qquad (1)$$

donde $\Omega_\mathrm{i} = \Omega_\mathrm{i}(f(\vec{x},t))$ es un operador de colisión. $\Delta \mathrm{t}$ y $\overrightarrow{\Delta \mathrm{x}}$ son los incrementos discretos de tiempo y espacio, respectivamente.

Empezando desde un estado inicial, la población de partículas en cada paso de tiempo se puede dividir en dos subpasos consecutivos:

Collision: $\tilde{f}_i(\vec{x},t) = f_i(\vec{x},t) + \Omega_\mathrm{i}$, interacción entre partículas de un nodo de acuerdo a reglas de dispersión.

Streaming: $f_i\left(\vec{x} + \mathrm{e_i}\overrightarrow{\Delta \mathrm{x}}, t + \Delta \mathrm{t}\right) = \tilde{f}_i(\vec{x},t)$, movimiento de cada partícula al siguiente nodo según su velocidad.

donde $\tilde{f}_i(\vec{x},t)$ representa el estado post-colisión.

La densidad ρ y el momento ρu son definidos como momentos de la velocidad de partícula de la función de distribución, f_i,

$$\rho = \sum_{i=1}^{M} f_i \qquad (2)$$

$$\rho u = \sum_{i=1}^{M} f_i e_i \qquad (3)$$

La naturaleza simple del LBM y su estructura de malla con reglas y operaciones locales posibilita una simple implementación paralelizada y adaptable a geometrías con fronteras complejas (Harting, et. al.). Los modelos de LB, se pueden diferenciar según la grilla y el operador de colisión utilizado.

2.1 Grillas

Las diferentes estructuras de grilla utilizadas, también llamados *lattice models* se refieren usualmente con la convención DdQq, donde d representa la dimensión espacial y q la cantidad de vecinos por celda (Qian et. al., 1992). Los modelos de uso más extensivo se pueden visualizar en la Figura 1.

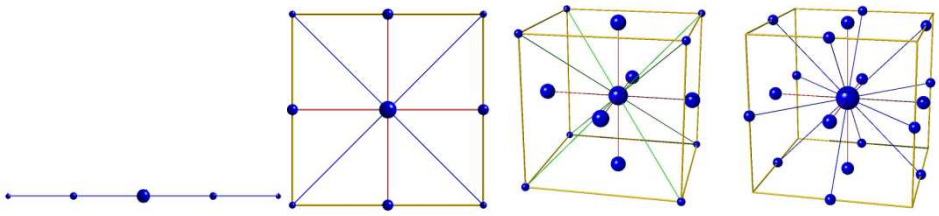

Figura 1: Distribución de partículas en las *lattices* en más comunes. De izquierda a derecha: D1Q5, D2Q9, D3Q15, D3Q19

Un análisis extensivo de los diferentes modelos de grillas y las consecuencias de su uso en cuanto a estabilidad numérica y la correcta recuperación de las ecuaciones del modelo puede verse en el Chikatamarla (2009).

2.2 Operadores de colisión

El LBM surgió como una solución para la simulación de fluidos que siguen las ecuaciones de Navier-Stokes. Luego se desarrollaron diferentes variantes para resolver otros modelos físicos, los cuales sólo difieren en el operador de colisión usado. En todos los casos el operador de colisión debe conservar las propiedades macroscópicas correspondientes al modelo físico representado.

2.2.1 Operadores SRT y MRT

En los operadores *Single Relaxation Time* (SRT) (Wolf-Gladrow, 2000), también llamados BGK (Bhatnagar et al., 1954), el operador de colisión está dado por

$$\Omega_i = \frac{f_i - f_i^{(eq)}}{\tau} \qquad (i = 0, 1, \dots, M) \qquad (4)$$

donde τ es el tiempo de relajación, y $f^{(eq)}$ es la distribución de equilibrio definida como una función de las variables macroscópicas en una celda según el modelo físico simulado.

En el caso de las ecuaciones de Navier-Stokes, $f^{(eq)}$ queda expresado como:

$$f_i^{eq} = \begin{cases} \frac{4}{9}\rho * \left(1 - \frac{3}{2}\|v\|^2\right) & i = 0, \\ \frac{1}{9}\rho(1 + 3 * \langle e_i, v_i\rangle + \frac{9}{2}\langle e_i, v_i\rangle^2 - \frac{3}{2}\|v\|^2 & i = 1,3,5,7, \\ \frac{1}{36}\rho(1 + 3 * \langle e_i, v_i\rangle + \frac{9}{2}\langle e_i, v_i\rangle^2 - \frac{3}{2}\|v\|^2 & i = 2,4,6,8, \end{cases} \qquad (5)$$

Existen diferentes funciones de equilibrio para cada modelo físico. Ejemplos de ello se puede encontrar en la resolución de las ecuaciones de Shallow Water para flujos de agua poco profundas (Zhou, 2004), los cuales se pueden utilizar para simulaciones de aguas de navegación en tiempo real (Garcia Bauza, 2010).

2.2.2 Simulación de iluminación

En el trabajo de Geist. et al., 2004 se planteó un modelo de iluminación basado en el LBM, que simula el transporte de fotones a través de un medio traslúcido. Para este sistema la regla de LBM es lineal, y se escribe como:

$$f_i\left(\vec{x} + \mathrm{e_i}\overrightarrow{\Delta \mathrm{x}}, t + \Delta \mathrm{t}\right) = M(\vec{x}, t) f_i(\vec{x}, t) \qquad (6)$$

donde $M(\vec{x}, t)$ es una matriz de colisión, cuyo elemento genérico $M_{i,j}(\vec{x}, t)$ es la probabilidad de que un fotón que entra con dirección e_i a la celda $\vec{x}$ en el tiempo t, salga con dirección e_j.

En Bulant y Maso (2010) se utiliza este modelo para *renderizar* escenas con cuerpos traslúcidos, como nubes. A su vez, se analiza la factibilidad del método para la resolución de tomografías difusiva óptica, las cuales se realizan con luz NIR (cercano al infrarojo) en lugar de rayos X.

2.3 Otras funcionalidades

Se han desarrollado numerosas variantes del método LBM que permiten incorporar nuevas características, como soporte de bordes, fuentes de masa y momento, fronteras inmersas móviles, etc. Para cada una de ellas existen diferentes variantes, con sus respectivas ventajas y desventajas.

2.3.1 No-slip

Las condiciones de contorno no-slip generan una rebote completo de todas las partículas que llegan a ella. La regla básica resultante es muy simple y establece que una partícula que alcanza el borde es revertida inmediatamente hacia el fluido, volviendo al mismo en el siguiente tiempo de simulación. Para implementar este método se utiliza el esquema *on-grid bounce-back* (Inamuro et al., 1995; Succi, 2001). Por otro lado existe el esquema del tipo *mid-grid* no-slip (He et al., 1997) en el cual las partículas rebotan en el punto medio entre la celda que todavía corresponde al fluido y la celda borde. Este esquema logra una aproximación de segundo orden de las ecuaciones de Navier Stokes.

2.3.2 Fuentes de masa y momento

Otra característica que presentan algunos motores de LBM es la capacidad de incorporar fuentes de masa y de momento (fuerzas). Para estas características existen diferentes versiones de aplicaciones de fuerza. Algunas de ellas actúan como parte del operador de colisión modificando la velocidad en el cálculo de la distribución de equilibrio y otras pueden ser agregadas como un subpaso adicional (Zhow,2004).

3 ANALISIS Y DISEÑO DE UNA CAPA DE ABSTRACCIÓN PARA MOTORES DE LBM

Como el LBM ha evolucionado ampliamente en el campo científico han surgido una gran variedad de implementaciones del mismo. En general, las aplicaciones disponibles cuentan con un motor básico representando un modelo específico (Navier-Stokes, flujo bifásico, etc.) y algunas características avanzadas de interés para el desarrollador. Hay implementaciones especiales paralelizadas en grid-computing (Heuveline y Latt, 2007) (Harting et al., 2005), en placas gráficas (Rinaldi, 2011), etc. Si bien existen implementaciones que buscan ser modulares y flexibles, ofreciendo una interfaz orientada a objetos lo suficientemente útil sin sacrificar *performance* (OpenLB, Palabos), éstas suelen carecer de portabilidad, son difíciles de incorporar a una aplicación más grande, no proveen una interfaz clara, y generalmente utilizan diferentes nomenclaturas. A su vez, el costo de cambio de un motor a otro es grande ya que el código utilizado depende en gran medida de sus interfaces.

Los motores de LBM procedurales tienen problemas de rigidez debido a su implementación. Estos motores suelen aplicarse a dominios con celdas uniformes y generalmente utilizan la memoria de manera poco eficiente, en particular cuando se requiere dar un tratamiento especial a celdas particulares (bordes, obstáculos, etc.). Por su parte, las implementaciones orientadas a objetos proveen clases que se pueden asignar a cada celda, permitiendo dominios mixtos.

La biblioteca OpenLB tiene el problema de que sólo permite advección a los nodos más cercanos. Las implementaciones en CUDA disponibles en general están optimizadas para problemas particulares, por lo que el dominio de aplicación es muy restringido y presentan poca flexibilidad a las modificaciones debido a la importancia dada a la performance. Estas bibliotecas presentan una serie de limitaciones generando una competencia entre la modificabilidad del código y la performance de ejecución del motor. En otras palabras, incorporar nuevas capacidades a un motor hace que crezca el tamaño del código, lo cual

dificulta el mantenimiento de una buena modularización. El mayor problema para disponer de motores eficientes es evitar que se calculen partes no utilizadas en la simulación, lo cual puede conducir a construir diversos motores o a construir un motor con una gran cantidad de opciones de configuración.

La configuración de experimentos suele utilizar extensos códigos de configuración, dependientes de la interfaz del motor, y con configuraciones específicas para cada experimento, por lo que es difícil la reutilización. Dichos códigos quedan atados a la interfaz del motor, por lo que ante un cambio de motor será necesario adaptar el experimento a la nueva interfaz. Lo mismo ocurre con las interfaces gráficas implementadas, en las cuales el volumen de código total es mucho mayor.

En este trabajo se propone utilizar una capa de abstracción para motores de *lattice* Boltzmann (CALB), la cual permite unificar la interfaz de los motores procedurales y orientados a objetos, independientemente de las características individuales que presenten. Esta característica es crucial, dado que se busca que la CALB sirva para abstraer motores de LBM que representen cualquier modelo físico, permitiendo así una mayor reutilización de código, especialmente de las interfaces gráficas y código de control de simulación. También se busca abstraer las dimensiones y modelos de grilla que se pueden utilizar. Por último, el mecanismo de abstracción permite acceder a cualquier funcionalidad disponible, o agregarla a otro motor de LBM que no lo tenga, manteniendo transparencia entre ambos casos a la hora de su uso.

3.1 Análisis

El objetivo de la capa de abstracción es proveer un acceso uniforme a cualquier motor de LBM. Para ello primero es necesario disponer de algún mecanismo para identificar las capacidades soportadas por cada motor. A su vez, para agregar desde la capa nuevas capacidades a un motor ya existente, debe abstraerse el comportamiento común de todos los motores de LBM en un conjunto reducido de clases; luego las clases adicionales se usarán para cada funcionalidad ausente. Lo común en cualquier motor LBM es la representación básica mediante poblaciones de partículas en cada celda clasificadas en un conjunto finito de velocidades y la regla de actualización colisión-advección. La capa debe proveer clases para acceder a las variables macroscópicas que funcionen con la mayoría de los motores y que puedan ser fácilmente especializadas.

3.2 Diseño

A continuación se propone un diseño de clases, detallando para cada clase funcionalidad y los principales métodos. En la Figura 2 se muestra el diagrama de clases de CALB mostrando funcionalidades e implementaciones externas de las mismas. Existe una clase principal denominada ***AbstractBoltzmann*** que provee la interfaz común a todos los motores. Cualquier otra funcionalidad provista se almacenará mediante un objeto apropiado en una tabla de hash de funcionalidades. La capa define las interfaces para manipular cada configuración que permitan los diferentes motores.

3.2.1 Clases para abstraer motores de LBM:

LatticeModel

Generaliza los distintos esquemas para la vecindad entre celdas, permite obtener la posición de cualquier celda vecina, sin importar la cantidad de dimensiones ni la cantidad de velocidades del modelo (D*dimension*Q*velocidades*).

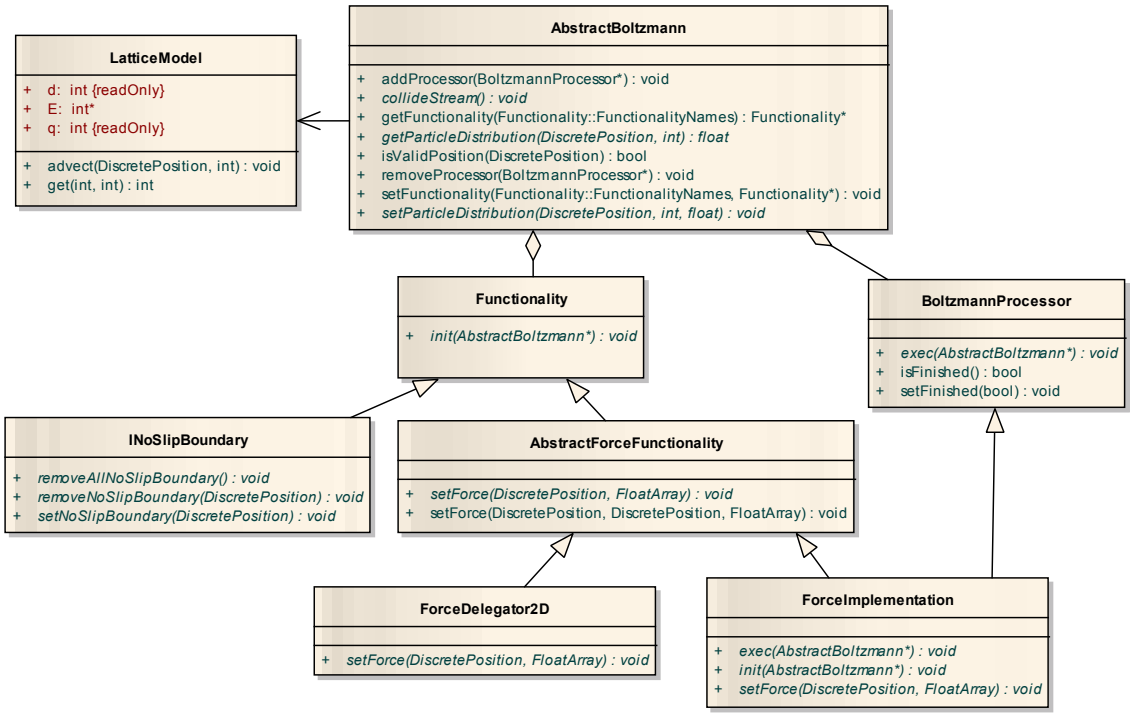

Figura 2: Diagrama de clases de la CALB mostrando funcionalidades e implementaciones externas de las mismas (como ejemplo se usa la funcionalidad de fuerza)

AbstractBoltzmann

Clase principal que encapsula el LBM utilizado. Define métodos abstractos para las características comunes (***collideStream()***, ***getParticleDistribution()*** y ***setParticleDistribution()***). La referencia a una posición de la distribución de partículas se generaliza con el par <CELDA, VELOCIDAD>. El método ***getFunctionality()*** recupera los objetos que permiten delegar funcionalidades específicas al motor. En caso que la funcionalidad consultada no sea soportada por el motor se devolverá ***NULL***.

Functionality

Cada interfaz de funcionalidad específica debe extender esta interfaz, siguiendo el patrón *Marker Interface* (Gamma et. al., 1995). Provee un método ***init()*** que en caso de ser necesario podrá actuar sobre el autómata.

AbstractForceFunctionality y NoSlipBoundary

Definen los métodos a utilizar para configurar una fuerza y para agregar-quitar bordes sin desplazamiento. Las implementaciones concretas que deleguen a un motor deben extender de ellas. A su vez, en estas clases se pueden definir métodos *template* que den mayor facilidad de uso para el programador (como aplicar una fuerza en un área rectangular o setear en 0 la fuerza en todo el dominio). Ejemplifican funcionalidades no soportadas por todos los motores.

BoltzmannProcessor

Toda funcionalidad agregada necesitará realizar procesamiento como parte del método ***computationalStep()***. Se utiliza el patrón ***Command*** (Gamma et. al., 1995), para permitir incorporar procesamiento en el mismo. Toda capacidad agregada debe utilizar los mismos mecanismos de acceso que aquellas que le son delegadas al motor. Para esto implementa la interfaz que define los métodos de la funcionalidad.

BoltzmannRunner, BoltzmannReader y BoltzmannCheck

BoltzmannRunner contiene un *thread* con un bucle interno. La condición de corte consiste en un ***BoltzmannCheck*** determinado por el usuario. De esta forma se pueden ejecutar autómatas fácilmente en un *thread* independiente. Por otra parte, luego de realizar cada paso de simulación se notifica a los lectores (***BoltzmannReader***) de manera asincrónica. Para configurar una simulación con eventos durante su ejecución se pueden agregar procesadores al autómata, permitiendo así tener una lista de eventos, de los que se debe manejar el momento a ser aplicados. La diferencia conceptual que existe entre un ***BoltzmannReader*** y un ***BoltzmannProcessor*** está en que el primero sólo realiza lecturas por lo que pueden ejecutarse todos en paralelo, mientras los procesadores se ejecutan secuencialmente ya que pueden realizar escritura de datos. Estas clases forman un *framework* que permite la fácil implementación de experimentos y aplicaciones en un esquema orientado a objetos. La relación entre todas estas clases se puede ver en el diagrama de clases de la Figura 3.

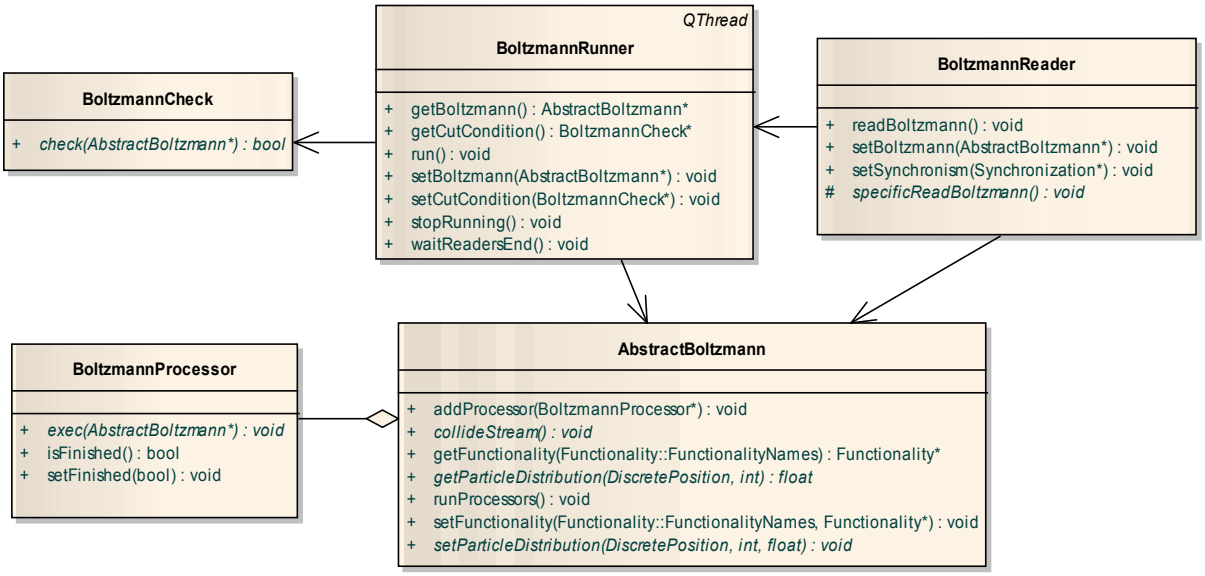

Figura 3: Diagrama de clases de una simulación según CALB.

3.1 Incorporación de un motor a la CALB

En Dottori, 2011 se dedica un capítulo a la incorporación de un nuevo motor a CALB, siguiendo paso a paso ejemplos simples, y detallando un proceso iterativo e incremental.

4 RESULTADOS

Para analizar la CALB se desarrolló una aplicación que permiten visualizar las variables macroscópicas en tiempo real y guardar salidas en formato de archivo, incluyendo estados estacionarios y transitorios de simulación. Los resultados sugieren que CALB puede ser muy útil en las primeras etapas de desarrollo de un motor o un experimento, ya que además permite visualizar rápidamente las diferentes variables macroscópicas para verificar el correcto funcionamiento.

4.1 Overhead introducido por la CALB

Al desarrollar una capa de abstracción, uno de los parámetros decisivos que determinan su utilidad es la penalización en tiempo extra de ejecución (*overhead*) que introduce dicha capa. En este estudio se calculó el overhead por iteración introducido por la capa en tres motores

diferentes. En la Fig. 4 se muestran los resultados obtenidos, los cuales indican que los tiempos de *overhead* son bastantes bajos. Cabe señalar que el *overhead* introducido en la ejecución es compensado con las facilidades brindadas por la capa, por ejemplo la disminución del tiempo de desarrollo de una aplicación.

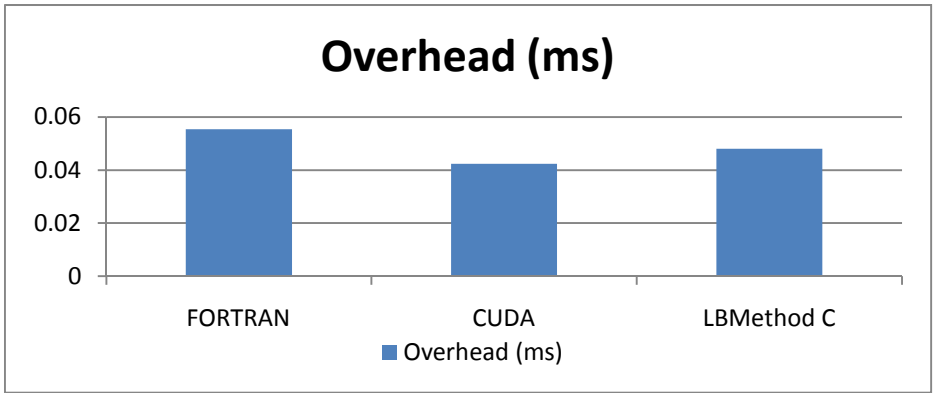

Figura 4: Overhead promedio por iteración [ms] introducido por la CALB.

4.2 Costo de desarrollo de una aplicación

Como la disminución del costo de desarrollo es la principal ventaja de CALB se realizaron ensayos para determinarlo en forma cuantitativa. Se eligió una simulación simple que consiste en ejecutar un número de iteraciones sin ningún tipo de configuración adicional a la provista por defecto en los motores. Para medir el costo de desarrollo de cada una de las versiones se utilizó la métrica *Source Lines of Code* (SLOC) (Putnam, 1978; Boehm, 1981).

	D2Q9 FORTRAN	D2Q9 CUDA	LBMethod C	CALB
SLOC	27	12	20	4

Tabla 1: Cantidad de líneas de código para una simulación simple sobre diferentes motores.

A partir de la Tabla 1 se observa como la CALB encapsula y abstrae al usuario de las particularidades de cada motor, otorgando valores por defecto a cada variable de inicialización. Cada variable puede ser configurada en caso de ser necesario. A su vez, cada aplicación desarrollada con la CALB puede ser utilizada con cualquiera de los motores cambiando solamente una línea de inicialización del motor.

4.3 Balance de la conveniencia del uso de las funcionalidades implementadas en CALB

Dado que el uso de CALB implica un balance entre la penalización en tiempo de ejecución y la ganancia en facilidad de desarrollo, cuando una funcionalidad determinada no es provista por el motor deseado, para el usuario se plantea el dilema de utilizar la implementación provista por CALB o modificar el motor para que soporte esta funcionalidad nativamente. La implementación dentro del motor puede ser más eficiente, pero con un costo de implementación mucho mayor. La respuesta a esta pregunta dependerá en gran medida de la experiencia del programador con el motor en particular. Para poder elegir a priori entre una y otra implementación se deben estimar estos parámetros y luego utilizar un criterio de decisión entre ambas.

Para analizar la competencia entre modificabilidad y performance se propuso el siguiente caso de estudio. Se parte de un motor implementado en C que está soportado por la capa de

abstracción. Dicho motor utiliza un operador de colisión BGK, sin soportar funcionalidades como bordes, celdas porosas y fuerzas. Estas tres funcionalidades se implementan tanto en la CALB como en diferentes versiones del motor. Para medir la modificabilidad y performance se definen las siguientes métricas:

- Costo de implementación (S): líneas de código agregadas para soportar una funcionalidad / total de líneas del motor original.
- Performance (C): tiempo por iteración (en ms)

	En el motor		En la CALB	
Casos de uso típicos	S	C	S	C
Motor simple	0,00	1,87	0,00	1,87
Bordes	0,08	1,9	0,01	2
Fuerzas	0,21	2,1	0,01	3
Bordes y fuerzas	0,29	2,14	0,02	3,25
Porosidad	0,14	2	0,01	4,9
Porosidad y fuerza	0,36	2,19	0,02	6,3
Porosidad, fuerza y bordes	0,43	2,24	0,03	6,7

Tabla 2: Mediciones de costo de desarrollo y performance tomadas para distintos casos típicos de motores.

En la Tabla 2 se detallan los resultados del estudio, donde se puede comprobar que como era de esperar la implementación directa de las funcionalidades bordes, fuerzas y porosidad es más eficiente desde el punto de vista del cálculo pero requiere mayor esfuerzo de desarrollo. La elección de uno u otro caso dependerá de la importancia relativa que le asigne el programador a cada métrica. Esta clase de decisión se resuelve comúnmente mediante el uso de una función de utilidad que representa la utilidad subjetiva desde el punto de vista del usuario para un par (C, S) dado. En el presente análisis, la función de utilidad es la planteada similar a la utilizada en microeconomía (Henderson y Quandt, 1980):

$$U = \exp(-S - \alpha C) \tag{7}$$

Donde α representa la importancia relativa de la performance respecto del costo de implementación. La mejor alternativa está dada según el esquema que provea mayor valor de utilidad U. A mayor alfa, mayor es la importancia que el usuario le otorga a la performance. En la Figura 5 se muestran los valores de alfa hasta los cuales es conveniente utilizar la CALB en cada una de estas funcionalidades.

Se puede observar que en las funcionalidades de fuerzas y porosidad la penalización en performance es mayor debido a que requieren cálculo sobre todo el dominio. El cálculo de porosidad tiene una gran diferencia de performance debido al costo de acceso a toda la memoria a través de la CALB. Por otra parte, cada funcionalidad que se implementa a nivel de capa se puede utilizar con diferentes motores, favoreciendo la reutilización en distintas implementaciones.

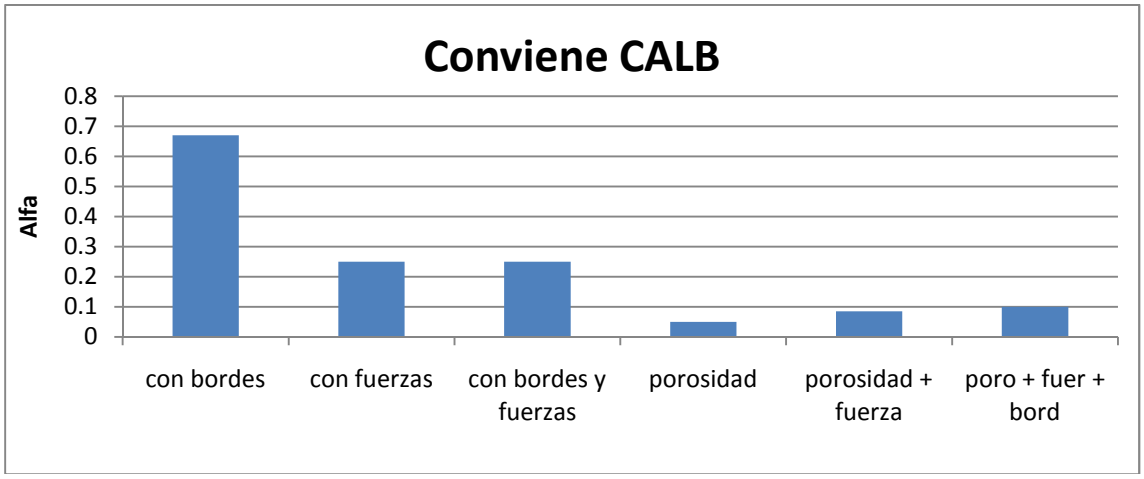

Figura 5: Valor de α por debajo del cual conviene usar CALB.

5 CONCLUSIONES

En el presente trabajo se implementó una solución que permite abstraer diferentes motores de LBM. La misma consiste en una capa de abstracción (CALB) que oculta las diferencias entre los motores. Una característica importante de CALB es el mecanismo unificado para acceder a la funcionalidad específica. Además se permite agregar nueva funcionalidad a un motor ya existente sobre CALB.

Se desarrolló un mecanismo que permite la configuración de simulaciones completas en un esquema orientado a objetos. CALB permite utilizar el código de usuario desarrollado con nuevos motores. Un claro ejemplo de esta característica es la aplicación desarrollada sobre la capa. La misma provee una visualización del fluido que puede ser utilizada con cualquier motor disponible en CALB.

Un usuario que haga uso de CALB puede comparar distintos motores modificando muy pocas líneas de código permitiendo así elegir aquel que mejor se adapte a sus necesidades. El *overhead* introducido resulta aceptable como se comprobó en los experimentos realizados. A su vez CALB facilita el desarrollo de la aplicación al proveer una interfaz sencilla y uniforme.

Si bien se logró incorporar diferentes motores a CALB, incluyendo uno que utiliza la placa gráfica para su procesamiento, existe la posibilidad de dar soporte a motores que distribuyan el cómputo sobre procesadores distribuidos. Esto incluiría dar acceso a la configuración utilizada para la comunicación y la distribución de trabajo entre los distintos procesadores.

REFERENCIAS

Bhatnagar, P. L., Gross, E. P. & M. Krook. A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems. *Phys. Rev., vol. 94*, pp. 511–525, 1954.

Bulant, C. A., Maso Talou, G. Autómatas de Lattice Boltzmann para modelos de iluminación difusa. *Tesis de grado en Ingeniería en Sistemas, UNCPBA*, 2010.

Chen, S. & Doolen, G. D. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid* Mechanics, 30(1):329-364, 1998.

Cheng, Y. & Zhang, H. Inmersed boundary method and lattice Boltzmann method coupled FSI simulation of mitral leaflet flow. *Computer & Fluids*, 39:871-881, 2010.

Chikatamarla, S., & Karlin, I. Lattices for the lattice Boltzmann method. *Physical Review* E, 79(4), 1-18. doi: 10.1103/PhysRevE.79.046701, 2009.

Dottori, J. & de la Fuente, E. CALB: Una capa de abstracción para motores de lattice Boltzmann. *Trabajo final de carrera Ingeniería en Sistemas*, UNCPBA, 2011.

García Bauza, C., Boroni, G., Vénere, M. & Clausse, A. Real-time interactive animations of liquid surfaces with Lattice-Boltzmann engines. *Aust. J. Basic & appl. Sci.* (ISI). ISSN 1991-8178, p. 3730-3740, 2010.

Geist, R., Rasche, K., Westall, J. & Schalkoff, R. Lattice-Boltzmann Lighting. *Eurographics Symposium on Rendering*. H.W. Jensen, A. Keller (Editors). Clemson University, Clemson, South Carolina USA 29634, 2004.

He, X., Zou, Q., Luo, L.-shi, & Dembo, M. Analytic Solutions of Simple Flows and Analysis of Nonslip Boundary Conditions for the Lattice Boltzmann BGK Model, *87*(Xcm), 115-136, 1997.

Henderson, J. M. y Quandt R. E. Microeconomic Theory. *McGraw-Hill*, New York, USA, 1980.

Inamuro, T., Yoshina, M., & Ogino, F. A non-slip boundary condition for lattice Boltzmann simulations. Phys. Fluids, 7:2928–2930, 1995.

Kadanoff L. On two levels. Phys. Today 39:7–9, 1986.

LBMethod.org. Sitio dedicado al LBM y su teoría. http://www.lbmethod.org/openlb/lb.theory.html

Rinaldi, P.R., Dari, E., Vénere, M.J., Clausse, A. Uso de GPUs para la Simulación de Fluidos en 3D con el Método de Lattice Boltzmann. *MECOM* (IX, Buenos Aires, Argentina). Mecánica Computacional, Vol. XXIX; Buenos Aires, Argentina: Dvorkin, E., Goldschmit, M., Storti, M. (eds.), 2010. pp. 7095-7108, 2010.

Rinaldi, P. R. Modelos de autómatas celulares sobre unidades de procesamiento gráfico de alta performance. Tesis Carrera de Doctorado en Ciencias de la Ingeniería, Director: Dr. Enzo Alberto Dari, Codirector: Dr. Marcelo Javier Vénere. Marzo de 2011.

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

Wolf-Gladrow, D. A. Lattice gas cellular automata and lattice Boltzmann models: an introduction. *Lecture notes in mathematics*, Springer, 2000.

Zhou J. G. Lattice Boltzmann methods for Shallow Water Flows. *Springer-Verlag*, 2004.