

INF-351: Computación de Alto Desempeño

Laboratorio 2 y 3

LGA en GPU

Prof. Álvaro Salinas

23 de Enero de 2020

1. Descripción y Marco Teórico

En el siguiente laboratorio serán evaluados sus conocimientos sobre divergencia, uso de memoria global y accesos de memoria coalescentes. Adicionalmente, usted deberá demostrar un correcto manejo de la herramienta NVIDIA Visual Profiler.

Lattice Gas Automata

Lattice Gas Automata (LGA) es un tipo de autómatas celular utilizado para realizar simulaciones de fluidos a través de la resolución de las conocidas ecuaciones de Navier-Stokes. Este método es el predecesor del actual ampliamente utilizado método de lattice Boltzmann. Si bien la teoría de dinámica de fluidos relacionada a este método no será revisada debido a que no es relevante para el desarrollo de este laboratorio, sí es necesario que analicemos su mecánica para comprender su implementación.

En esta ocasión, implementaremos un tipo especial de LGA, denominado modelo HPP. Éste consiste en una malla uniforme equiespaciada conformada por nodos que se encuentran a una distancia de 1 lattice unit (lu). En cada nodo es posible tener hasta 4 partículas, cada una asociada a una dirección. Dichas partículas solo pueden existir en un nodo y son representadas por un valor binario $f_i(\mathbf{x}, t)$ que representa la presencia (1) o ausencia (0) de una partícula en la i -ésima dirección del nodo en la posición $\mathbf{x} = (x, y)$ en el tiempo t . La Figura 1 presenta el esquema recién descrito.

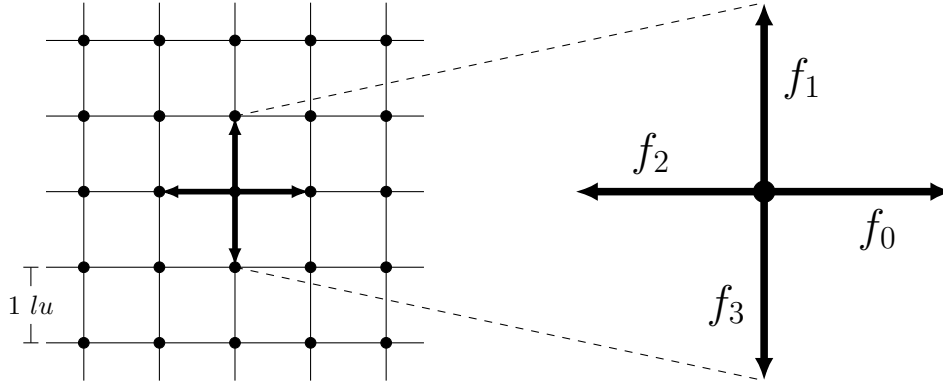


Figura 1: Esquema del modelo HPP.

El tiempo se discretiza en time steps (ts), i.e. $\Delta t = 1 ts$ con $t_i = t_{i-1} + \Delta t$. Un time step del modelo HPP consiste en dos pasos denominados collision y streaming. El primero de ellos consiste en la representación de las posibles colisiones que pueden generarse entre partículas que llegan a un

mismo punto desde direcciones opuestas, mientras que el paso de streaming modela el movimiento de las partículas desde un nodo a su vecino según la dirección de cada una de ellas.

Es importante notar que el paso de collision es un proceso local de cada nodo, ya que solo supone un reordenamiento de las partículas respecto a sus direcciones. Por otro lado, el paso de streaming no es un proceso local, pues existe interacción entre el nodo y su vecindad al trasladarse las partículas. En este contexto, solo son considerados dos casos posibles para el paso de collision, los cuales corresponden a la presencia de únicamente dos partículas en dirección opuesta, en cuyo caso deben ser rotadas en 90° . La Figura 2 contiene ejemplos ilustrativos de evolución en un time step (t_i a la izquierda y t_{i+1} a la derecha). Notar que el segundo y tercer ejemplo presentados en la figura son los únicos casos en los que hay collision.

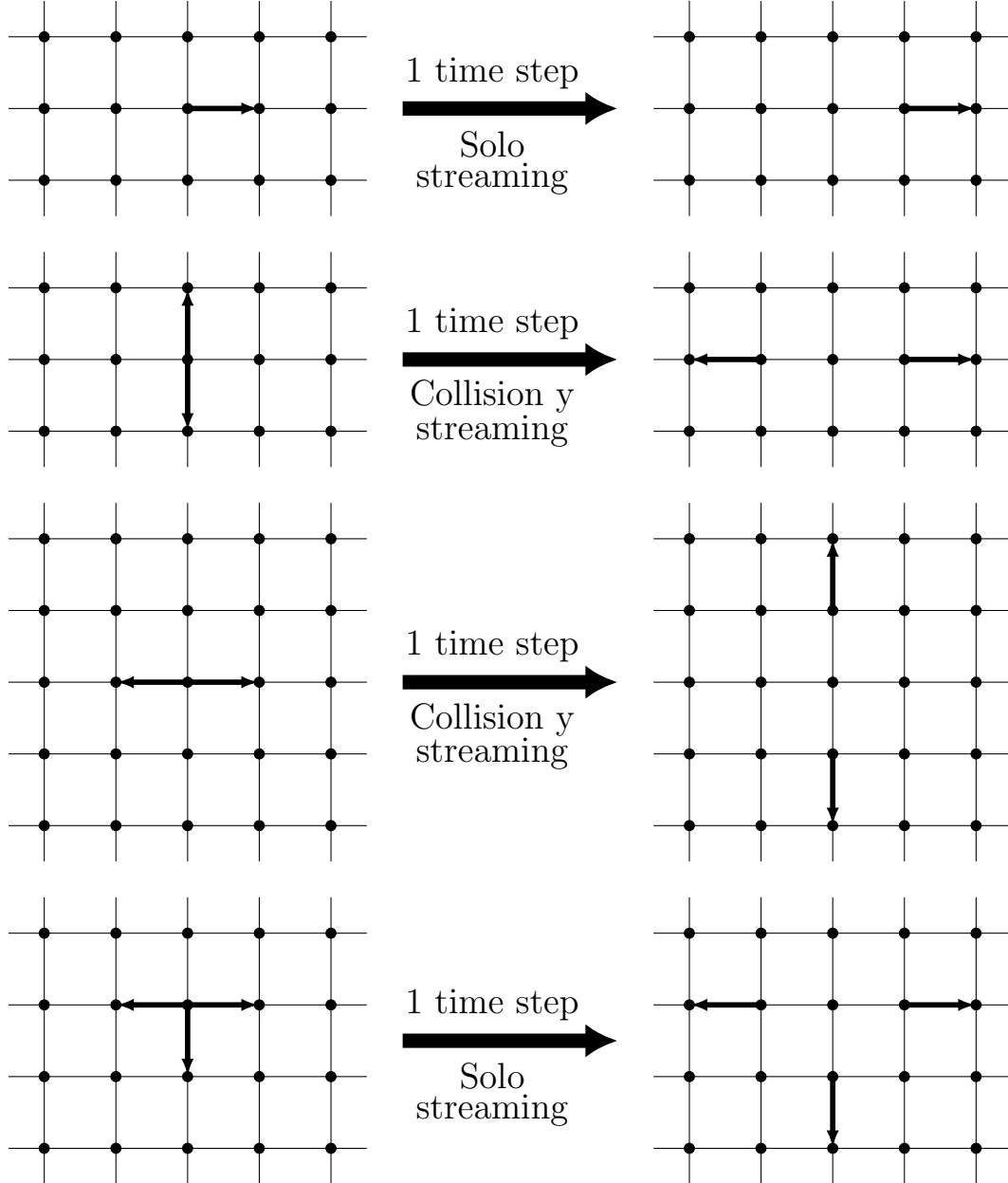


Figura 2: Evolución del método en un time step.
Solo son dibujadas las partículas existentes, i.e. $f_i = 1$.

2. Desarrollo

Para el desarrollo de este laboratorio usted deberá descargar el archivo `initial.txt` publicado justo a este enunciado. Este archivo contiene la distribución inicial de partículas con la cual debe comenzar su simulación. El formato de este archivo es:

- La primera línea contiene dos enteros N y M correspondientes a la dimensión (número de nodos) de la malla en la dirección y y x respectivamente.
- Las líneas 2, 3, 4 y 5 del archivo contienen cada una $N \times M$ valores binarios correspondientes a la presencia o ausencia de una partícula en una dirección de cada nodo. La i -ésima línea del archivo contiene los valores f_{i-2} (ver Figura 1), e.g. la línea 2 contiene los valores f_0 de cada nodo. Considere que la malla se representa mediante filas concatenadas, es decir, primero están los valores asociados a los M nodos con $y = 0$, después los M nodos con $y = 1$, y así sucesivamente hasta los M nodos con $y = N - 1$.

Adicionalmente, se le provee el archivo `script.ipynb`, el cual contiene un script para generar mallas de distintos tamaños y graficar los resultados. Si bien puede generar instancias más pequeñas con esta herramienta, procure que sus conclusiones y mediciones de tiempo sean en base al archivo con la data inicial entregado.

Para todas las implementaciones que realizará en este laboratorio, compruebe que se conserva la masa del sistema:

$$\sum_{k=1}^4 \sum_{j=1}^N \sum_{i=1}^M f_k((x_i, y_j), t_q) = \sum_{k=1}^4 \sum_{j=1}^N \sum_{i=1}^M f_k((x_i, y_j), 0) = \text{constante} \quad \forall q \in \{1, \dots, 1000\}$$

Esto significa que para todos los tiempos simulados, el número de partículas en el dominio se mantiene constante, ya que no deberían desaparecer partículas ni aparecer nuevas. Basta con que compruebe para el tiempo final y es libre de decidir si comprobarlo con código de GPU o CPU.

- a) Cree dos funciones de CPU encargadas de leer el archivo `initial.txt`, una utilizando el enfoque Structure of Arrays y la otra usando Array of Structures, obteniendo en ambos casos un array de dimensión $N \times M \times 4$. Para cada una de ellas, implemente los CUDA kernels que considere necesarios para simular un time step del método, considerando en todos los casos que cada hebra se encarga de procesar un nodo de la malla. Simule 1000 time steps llamando a los kernels desarrollados en un ciclo for de 1000 iteraciones. Realice un profiling de cada versión y obtenga el tiempo que tardaron las 1000 invocaciones en cada caso.

Para efectos de su implementación considere condiciones de borde periódicas, i.e. los nodos en el borde superior ($y = N - 1$) son vecinos de los nodos en el borde inferior ($y = 0$) y lo mismo ocurre para los nodos de las fronteras este ($x = M - 1$) y oeste ($x = 0$).

Hint: Recuerde que debe evitar las condiciones de carrera que pudiesen surgir en el paso de streaming, e.g. cuando una hebra intenta trasladar una de sus partículas a uno de sus vecinos sobreescribiendo su valor, pero la hebra encargada de dicho vecino aun no ha realizado este proceso.

[Pregunta] ¿Cuál de las dos implementaciones realizadas muestra un mejor desempeño? Comente al respecto.

- b) Implemente una tercera versión en la cual los cuatro valores binarios f_i asociados a cada nodo se almacenen en un único elemento de un array de dimensión $N \times M$. De esta forma, los cuatro valores binarios deben estar almacenados en 4 bits de cada elemento. Por ejemplo, si utilizamos los bits de derecha a izquierda, en caso de que $f_0 = 1$, $f_1 = 0$, $f_2 = 1$ y $f_3 = 1$, el valor almacenado en el arreglo es 13 debido a que los bits del último byte del elemento que contiene ese valor son 00001101. Usted tiene la libertad para elegir cuáles bits utilizar y en qué orden, pero debe ser consistente para todos los nodos. Ejecute también 1000 iteraciones.

Presente una tabla con los tiempos reportados por el profiler para esta implementación y las dos versiones de la pregunta a). Muestre también en la tabla el ancho de banda utilizado en los tres casos (use el ancho de banda Total reportado en Device Memory en el profiler).

[Pregunta] ¿Esta tercera implementación resultó más eficiente que las dos anteriores? ¿A qué se debe esto?

[Pregunta] ¿Resulta más conveniente trabajar directamente con el valor decimal almacenado en el array u obtener los 4 bits y almacenarlos en variables locales distintas? ¿Existe algún cambio en la ocupancia obtenida en cada caso?

- c) Elija la versión más eficiente de las tres implementadas y modifique el problema resuelto cambiando a condiciones de borde de muro. En este caso, las partículas que llegan a un nodo situado en el borde con una dirección ortogonal a dicho borde se reflejan o “rebotan” hacia el sentido contrario antes de realizar el streaming (ver Figura 3). Para esta implementación, omita el paso de collision en los nodos de borde, i.e. si existen únicamente dos partículas moviéndose en dirección opuesta en el mismo nodo ubicado en la frontera, permita que se “atravesen” y sigan su camino sin rotar en 90° como en el resto de los nodos. Compare la utilización de if-statements, operador ternario y multiplicación booleana en el manejo de los condicionales relacionados al paso de collision y las condiciones de borde (utilice el mismo método a la vez para verificar ambos tipos de condición). Realice un profiling de su implementación y reporte en una tabla los tiempos obtenidos para las 1000 iteraciones con cada método.

[Nota] El archivo con la data inicial fue generado aleatoriamente. Puede que en la primera iteración se pierdan partículas por el hecho de tener duplicados en un nodo de borde. Por ejemplo, si un nodo en un borde vertical (izquierda o derecha) comienza con $f_0 = 1$ y $f_2 = 1$, entonces una partícula rebotará y se moverá a la misma posición que la otra. Esto no supone ningún problema para la implementación. Solo se menciona para que no se extrañen si es que al sumar todas las partículas, la cantidad disminuye después de la primera iteración. No debería ocurrir para las siguientes.

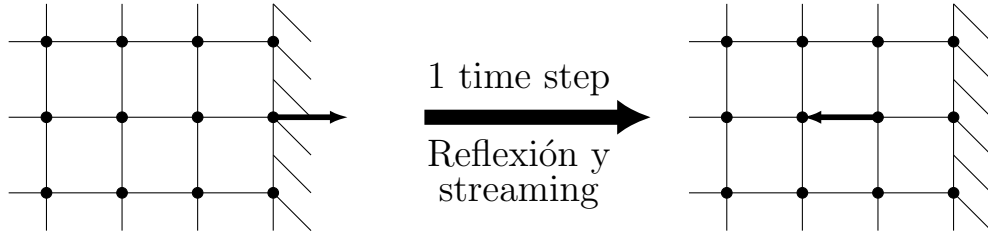


Figura 3: Condición de borde de muro.

[Pregunta] ¿Es alguno de los 3 métodos para manejar condiciones más eficiente que los demás? Comente el porqué de este resultado.

- d) Desarrolle este ítem y su correspondiente **[Pregunta]** solamente si sus implementaciones anteriores consistieron en más de un CUDA kernel.

También considerando condiciones de borde de muro, en esta ocasión, desarrolle un único CUDA kernel que simule un time step del método. En otras palabras, dentro del ciclo for no debiese existir ninguna otra instrucción que modifique o utilice los datos de las partículas aparte de la llamada al kernel mencionado. Puede realizar el preprocesamiento que desee a los datos antes de comenzar el ciclo.

Para la realización de este kernel utilice el método para manejar condiciones que obtuvo un mejor resultado en la pregunta anterior. Del mismo modo, siga utilizando la mejor distribución de los datos encontrada en las preguntas a) y b). Mida e indique el tiempo que demoran las 1000 iteraciones.

[Pregunta] ¿Resultó su nueva implementación ser más eficiente? ¿Aumentó, redujo o mantuvo el número de accesos a memoria global respecto a sus implementaciones anteriores? Comente al respecto.

3. Reglas y Consideraciones

Entrega

- La entrega debe realizarse en un archivo de nombre Lab2-X.tar.gz (formatos rar y zip también son aceptados), donde X debe ser reemplazado por el número de su grupo. Diríjase a la inscripción de grupos para consultar su número.
- El archivo de entrega debe contener un informe en formato pdf junto con el código implementado para resolver el laboratorio. Se le ruega entregar un código ordenado.
- El informe debe contener:
 - Título y número del laboratorio.
 - Nombre y rol de todos los integrantes del grupo.
 - Modelo y compute capability de la tarjeta gráfica que fue utilizada para ejecutar el código. Si se probó con más de una tarjeta gráfica, incluya los datos de todas y especifique qué tarjeta se utilizó en cada pregunta y resultado reportado.
 - Desarrollo. Preocúpese de responder cada pregunta señalada con el tag **[Pregunta]** en el enunciado.
 - Conclusiones. Incluya comentarios, observaciones o supuestos que surgieron durante el desarrollo del laboratorio.
- El descuento por día de retraso es de 30 puntos, con un máximo de 1 día de retraso. No se aceptarán entregas posteriores.
- En caso de copia, los grupos involucrados serán evaluados con nota 0. No hay problema en generar discusión y compartir ideas de implementación con sus compañeros, pero códigos copiados y pegados no serán aceptados.
- El no cumplimiento de estas reglas implica descuentos en su evaluación.
- La fecha de entrega es el día Viernes 14 de Febrero. Se habilitará la opción de entrega en aula.

Consideraciones

- Trabaje con arrays de enteros de 4 bytes (`int`) o de tipo `unsigned char` (1 byte).
- Para mediciones de tiempo utilice la herramienta Visual Profiler.
- Si lo desea, puede agregar capturas de pantalla del reporte del Visual Profiler. Éstas incluso pueden reemplazar las tablas solicitadas siempre y cuando se entienda claramente qué implementación está siendo analizada.
- Utilice 256 hebras por bloque en sus implementaciones.