



Universitat Politècnica de València

Master de Computación en la Nube y Altas Prestaciones

Conceptos y Métodos de la Computación Paralela

**Métodos estacionarios para la ecuación de
Poisson - Versión CUDA**

*Dawid Smalcuga
George-Alexandru Dogarel
Christian David León Ilaño*

Introducción

El objetivo de este proyecto es implementar una versión CUDA partiendo del código secuencial poisson.c. Se ha realizado una paralelización progresiva, en la que cada versión mejora la anterior, realizando un estudio comparativo de las prestaciones de cada versión.

Versión secuencial poisson.c

Análisis

- **Coste computacional secuencial** (para k iteraciones)

$$T_s = k \cdot 8 \cdot N \cdot M \text{ flops}$$

Tamaño	Tiempo (s)
600x600	162.26

- **Cuellos de botella de la versión secuencial**
 - Bucles anidados: Los bucles *for* en *jacobi_step* y *jacobi_poisson*
 - Acceso a memoria a los arrays *x*, *b* y *t*
 - Sincronización implícita: Cada iteración depende de la anterior.

1. Primera versión

En esta versión se paraleliza únicamente la función *jacobi_step* que se implementa con un kernel de CUDA que accede únicamente a la memoria global.

Análisis

- **Coste computacional paralelo**

$$T_p = k \cdot (5 \cdot N \cdot M) + t_c \text{ flops}$$

t_c es el tiempo de sincronización (*cudaDeviceSynchronize*) después de cada kernel y comunicación (*cudaMemcpy*), 3 copias por iteración: *x*, *b* y *t* al device.

$$T_c = 2 \times (N + 2) \times (M + 2) \times 8 \text{ bytes}$$

p es el número de hilos activos.

- **Overhead**

- Copia de datos entre host y device.
- Sincronización después de cada kernel.
- Cálculo del error en el host (requiere copiar t de vuelta al host).

Tamaño	BS	Tiempo (s)	SpeedUp
600x600	32	118.33	1.37
600x600	16	101.8	1.59
600x600	8	101.55	1.6
600x600	4	102.08	1.59
600x600	2	104.62	1.55

- **Strong scaling:** Si aumentamos el número de hilos (p), el tiempo aritmético disminuye, pero el tiempo de comunicación y sincronización puede convertirse en un cuello de botella. El algoritmo escalará bien si el tiempo de comunicación es pequeño comparado con el tiempo de cálculo.

2. Segunda versión

Esta versión optimizada del método de Jacobi utiliza **memoria compartida**.

Análisis

- **Overhead**
 - Sincronización global y dentro del bloque
- **Coste computacional paralelo**
 - Cálculo del nuevo valor t

$$5 \cdot N \cdot M$$

- Carga de datos en memoria compartida

$$\frac{N}{BS-2} \cdot \frac{M}{BS-2} \cdot BS \cdot BS = \left(\frac{BS}{BS-2}\right)^2 \cdot N \cdot M$$

- Reducción del error en el host

$$4 \cdot N \cdot M$$

$$T_{aritmético} = k \cdot \left(9 + \left(\frac{BS}{BS-2} \right)^2 \right) \cdot N \cdot M + t_c \text{ flops}$$

- **Coste de comunicación:** Mismo que en la Versión 1

$$T_c = 2 \cdot (N + 2) \cdot (M + 2) \cdot 8 \text{ bytes}$$

Tamaño	BS	Tiempo (s)	SpeedUp
600x600	32	119.11	1.36
600x600	16	102.09	1.59
600x600	8	101.55	1.6
600x600	4	104.73	1.55

3. Tercera versión: Atomic

Este código es una versión avanzada y optimizada del método de Jacobi. La principal mejora con respecto a las versiones anteriores es la reducción en paralelo del error, lo que elimina la necesidad de copiar los resultados de vuelta al host para calcular el error en cada iteración.

Análisis

- **Overhead**

- Sincronización global y dentro del bloque más reducción atómica

- **Coste computacional paralelo**

- Cálculo de nuevo valor de t

$$T_{nuevo\ valor} = 5 \cdot N \cdot M \text{ flops}$$

- Cálculo del error local

$$T_{error\ local} = 3 \cdot N \cdot M \text{ flops}$$

- Reducción del error dentro del bloque
 - Número de bloques

$$N\ blocks = \left[\frac{N}{BS-2} \right] \cdot \left[\frac{M}{BS-2} \right]$$

- Coste total

$$T_{reducción\ de\ bloque} = BS^2 + 1 \text{ flops}$$

$$T_{reducción\ bloques} = \left[\frac{N}{BS-2} \right] \cdot \left[\frac{M}{BS-2} \right] \cdot (BS^2 + 1) \text{ flops}$$

- Reducción global del error

$$T_{reducción\ global} = nBlocks \cdot (nBlocks + 1) \text{ flops}$$

- **Coste computacional aritmético total**

$$T_{total} = k \cdot \left(8 \cdot N \cdot M + \left[\frac{N}{BS-2} \right] \cdot \left[\frac{M}{BS-2} \right] \cdot (BS^2 + 1) + nBlocks \cdot (nBlocks + 1) \right) \text{ flops}$$

- **Coste de comunicación**

- Copias Device-Device: actualización de d_x al final de cada iteración.

$$T_{device-device} = k \cdot ((N + 2) \times (M + 2) \times 8) \text{ bytes}$$

- Copias Host-Device: copias de h_x y h_b al device al principio y otra al final.

$$T_{inicial} = 2 \times (N + 2) \times (M + 2) \times 8 \text{ bytes}$$

$$T_{final} = (N + 2) \times (M + 2) \times 8 \text{ bytes}$$

- Copias Device-Host: copia del error total al host en cada iteración.

$$T_{device-host} = 8 \cdot k \text{ bytes}$$

$$T_{c\ total} = 3 \times (N + 2) \times (M + 2) \times 8 + k \times 8 + k \cdot ((N + 2) \times (M + 2) \times 8) \text{ bytes}$$

Tamaño	BS	Tiempo (s)	SpeedUp
600x600	32	66.58	2.44
600x600	16	23.28	6.97
600x600	8	16.49	9.84
600x600	4	36.01	4.51

3. Tercera versión: Butterfly

Esta última versión incluye una reducción butterfly para el cálculo del error. Esto mejora el rendimiento al reducir el overhead de sincronización y conflictos de memoria.

Análisis

Los únicos cambios en el coste computacional respecto a la versión anterior son los siguientes:

- **Reducción del error dentro del bloque**
 - Cada bloque tiene BS^2 hilos.
 - La reducción butterfly requiere $\log_2(BS^2)$ pasos.
 - En cada paso, cada hilo realiza una suma

$$T_{reducción\ bloque} = \left[\frac{N}{BS-2} \right] \times \left[\frac{M}{BS-2} \right] \times BS^2 \cdot \log_2(BS^2) \text{ flops}$$

- **Reducción global del error**
 - Cada hilo realiza sumas en $\log_2(nBlocks)$ pasos.

$$T_{reducción\ global} = nBlocks \cdot \log_2(nBlocks) \text{ flops}$$

- **Coste aritmético total**

$$T_{total} = k \cdot \left(7 \cdot N \cdot M + \left[\frac{N}{BS-2} \right] \times \left[\frac{M}{BS-2} \right] \times BS^2 \cdot \log_2(BS^2) + nBlocks \cdot \log_2(nBlocks) \right) \text{ flops}$$

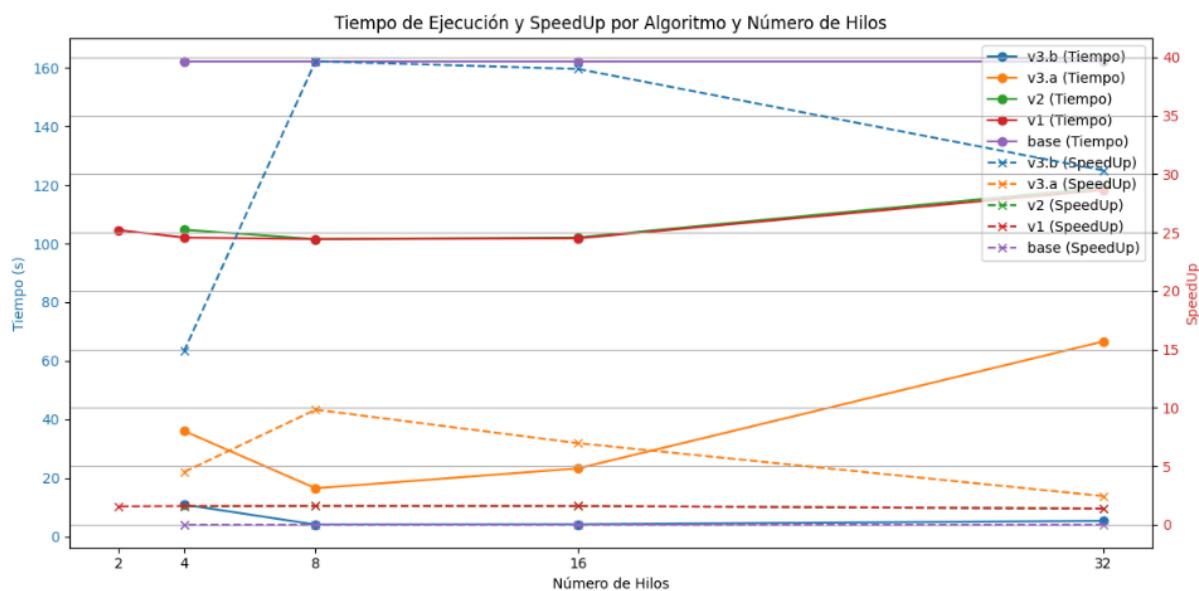
- **Tiempo de comunicación:** Mismo tiempo que en la versión anterior.

$$T_{c\ total} = 3 \times (N + 2) \times (M + 2) \times 8 + k \times 8 + k \times (N + 2) \times (M + 2) \times 8 \text{ bytes}$$

- **Overhead:** Esta versión requiere una sincronización explícita en cada paso y un menor overhead frente a la versión atómica ya que los hilos acceden a diferentes posiciones en cada paso.

Tamaño	BS	Tiempo (s)	SpeedUp
600x600	32	5.35	30.33
600x600	16	4.16	39.01
600x600	8	4.09	39.67
600x600	4	10.89	14.9

Comparación de versiones



Analizando la tabla mostrada con todos los tiempos y SpeedUps de las diferentes versiones en base a la cantidad de hilos de ejecución, podemos ver que las versiones 1 y 2 presentan una mejoría modesta debido al overhead de la comunicación y sincronización y que las versiones más eficientes son las de la versión 3. La versión atómica, gracias a la reducción del overhead de comunicación

y, la versión butterfly, que añade una mejor eficiencia en la reducción del error, es la versión más eficiente de todas.

Conclusiones

Hemos observado que con BS=32 los bloques tienen 1024 hilos y usan mucha memoria compartida y muchas atomicAdd hacia el mismo elemento compartido. Esto reduce la ocupación y crea una contención masiva en los atomicAdd, de modo que la GPU no puede ocultar latencias eficientemente.

Con BS=16 los bloques son más pequeños (256 hilos), usan mucho menos memoria compartida, hay muchos más bloques en la cuadrícula (más trabajo repartido) y menos contención en la reducción, por eso sale más rápido.

Este proyecto demuestra que la paralelización efectiva en CUDA requiere un enfoque iterativo, donde cada versión se construye sobre las lecciones aprendidas de la anterior. La versión 3: Butterfly es un ejemplo claro de cómo las optimizaciones en el acceso a memoria y la reducción del error pueden transformar un algoritmo, logrando un rendimiento excepcional. Para aplicaciones prácticas, esta versión es la más recomendable, especialmente cuando se busca maximizar el speedup con un número moderado de hilos.

Anexo 1: Propiedades del sistema

```
cdleoila@alumno.upv.es@bluenote:~/cmcp$ ./dquery
There are 2 devices supporting CUDA

Device 0: "NVIDIA RTX 4000 Ada Generation"
  Major revision number:          8
  Minor revision number:         9
  Total amount of global memory: 20991115264 bytes
  Number of multiprocessors:     48
MapSMtoCores for SM 8.9 is undefined. Default to use 64 Cores/SM
  Number of cores:               3072
  Total amount of constant memory: 65536 bytes
  Total amount of shared memory per block: 49152 bytes
  Total number of registers available per block: 65536
  Warp size:                     32
  Maximum number of threads per block: 1024
  Maximum sizes of each dimension of a block: 1024 x 1024 x 64
  Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
  Maximum memory pitch:           2147483647 bytes
  Texture alignment:              512 bytes
  Clock rate:                   2.17 GHz
  Concurrent copy and execution: Yes

Device 1: "NVIDIA RTX 4000 Ada Generation"
  Major revision number:          8
  Minor revision number:         9
  Total amount of global memory: 21000290304 bytes
  Number of multiprocessors:     48
MapSMtoCores for SM 8.9 is undefined. Default to use 64 Cores/SM
  Number of cores:               3072
  Total amount of constant memory: 65536 bytes
  Total amount of shared memory per block: 49152 bytes
  Total number of registers available per block: 65536
  Warp size:                     32
  Maximum number of threads per block: 1024
  Maximum sizes of each dimension of a block: 1024 x 1024 x 64
  Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
  Maximum memory pitch:           2147483647 bytes
  Texture alignment:              512 bytes
  Clock rate:                   2.17 GHz
  Concurrent copy and execution: Yes
```