Facultad de Ciencias Básicas

dm2a Doctorado en Modelamiento Matemático Aplicado

GEMA Grupo de Investigación Estudios en Matemáticas y Aplicaciones

High-Performance Computing para Modelamiento Matemático con (Py)CUDA

Sesión 2: Implementación paralela de métodos numéricos en GPU

Diego Maldonado

Universidad Católica del Maule

17 de diciembre de 2025

Índice

lectura_vector.py

```
1  import pycuda.autoinit
2  from pycuda.compiler import SourceModule
3  import pycuda.driver as drv
4  import numpy as np
5
6  mod = SourceModule(r"""
7  __global__ void LecturaVectorGPU(int n, double *A)
8  {
9    int i = threadIdx.x+blockIdx.x*blockDim.x;
10     printf("El hilo (%d) esta leyendo A[%d] = %f\n",i
         ,i,A[i]);
11 }
12 """)
13
14 LecturaVector = mod.get_function("LecturaVectorGPU")
15
16 A = np.array([1,2,3,4,5,6,7,8], dtype=np.float64)
17 n = len(A)
18
19 gdim = (1,1,1) # Dimensión de la grilla
20 bdim = (n,1,1) # Dimensión del bloque
21
22 LecturaVector(np.int32(n),drv.In(A),block=bdim,grid=
       gdim)
```

Salida:

```
El hilo (0) esta leyendo A[0] = 1
El hilo (1) esta leyendo A[1] = 2
El hilo (2) esta leyendo A[2] = 3
...
```

Nota

L3: Carga driver. Gestiona la memoria CUDA.

Ejercicio

- Cambie los valores de A.
- Cambie el tipo de dato de A a enteros.
- Cambie *n* en la linea 16 por 10.

Tipos de datos CUDA $\longleftrightarrow$ Python

CUDA	Python (NumPy)
`int`	`np.int32`
`unsigned int`	`np.uint32`
`float`	`np.float32`
`double`	`np.float64`
`long long`	`np.int64`
`char`	`np.int8`

Nota: al crear el arreglo en Python, especifique siempre el parámetro `dtype`, por ejemplo `arr=np.array([1,2,3],dtype=np.float32)`

lectura_vector.py

```
1  import pycuda.autoinit
2  from pycuda.compiler import SourceModule
3  import pycuda.driver as drv
4  import numpy as np
5
6  mod = SourceModule(r"""
7  __global__ void LecturaVectorGPU(int n, double *A)
8  {
9    int i = threadIdx.x+blockIdx.x*blockDim.x;
10     printf("El hilo (%d) esta leyendo A[%d] = %f\n",i
         ,i,A[i]);
11 }
12 """)
13
14 LecturaVector = mod.get_function("LecturaVectorGPU")
15
16 A = np.array([1,2,3,4,5,6,7,8], dtype=np.float64)
17 n = len(A)
18
19 gdim = (1,1,1) # Dimensión de la grilla
20 bdim = (n,1,1) # Dimensión del bloque
21
22 LecturaVector(np.int32(n),drv.In(A),block=bdim,grid=
       gdim)
```

Salida:

```
El hilo (0) esta leyendo A[0] = 1
El hilo (1) esta leyendo A[1] = 2
El hilo (2) esta leyendo A[2] = 3
...
```

Nota

- L3: Carga driver. Gestiona la memoria CUDA.
- L13: `drv.In(A)` define al arreglo `A` como entrada en `LecturaVector`.

Ejercicio

- Cambie los valores de A.
- Cambie el tipo de dato de A a enteros.
- Cambie *n* en la linea 16 por 10.

Tipos de datos CUDA ⟷ Python

CUDA	Python (NumPy)
`int`	`np.int32`
`unsigned int`	`np.uint32`
`float`	`np.float32`
`double`	`np.float64`
`long long`	`np.int64`
`char`	`np.int8`

Nota: al crear el arreglo en Python, especifique siempre el parámetro `dtype`, por ejemplo
`arr=np.array([1,2,3],dtype=np.float32)`

Lectura y escritura de arreglos

escritura_vector.py

```
1  import pycuda.autoinit
2  from pycuda.compiler import SourceModule
3  import pycuda.driver as drv
4  import numpy as np
5
6  mod = SourceModule(r"""
7  __global__ void escrituraGPU(int n, int *A){
8    int i = threadIdx.x+blockIdx.x*blockDim.x;
9    if (i < n) {
10     A[i]=i;
11     printf("El hilo (%d) esta escribiendo A[%d] = %d
        \n",i,i,A[i]);
12   }
13 }
14 """)
15
16 A = np.array([0,0,0,0,0], dtype=np.int32)
17 n = len(A)
18
19 EscrituraVector = mod.get_function("escrituraGPU")
20
21 gdim = (1,1,1) # Dimensión de la grilla
22 bdim = (n,1,1) # Dimensión del bloque
23
24 EscrituraVector(np.int32(n),drv.Out(A),block=bdim,
        grid=gdim)
25 print(A)
```

Salida:

```
El hilo (0) esta escribiendo A[0] = 0
El hilo (1) esta escribiendo A[1] = 1
...
[0,1,2,3,4]
```

Nota

L17: `dtype=np.int32` define el arreglo como entero de 32 bit.

L26: `drv.Out(A)` define al arreglo `A` como salida en `EscrituraVector`.

Ejercicio

- Cambie los valores de A.
- Cambie el tipo de dato de A a flotante.
- Cambie `A[i]=i` por `A[i]=10` en la linea 12.

1. Cree un programa que reciba como entrada un arreglo A de tamaño n de enteros y entregue como salida un arreglo B de tamaño n donde cada entrada es el doble que su contraparte en A. Imprima A y B.
2. Cree un programa que reciba como entrada un arreglo A de tamaño n de enteros y un entero k entregue como salida un arreglo B de tamaño n donde cada entrada es su contraparte en A multiplicada por k. Imprima A y B.
3. Modifique el programa anterior para recibir flotantes de 64 bits.
4. Modifique el programa anterior para entregar la salida en A mediante el comando `drv.InOut()`. Imprima A antes y despues de modificarlo.
5. Cree un programa que reciba como entrada dos arreglos A y B, flotantes de tamaño n y entregue un arreglo C que es la suma de ambos componente a componente. Imprima C.
6. Cree un programa que reciba como entrada dos arreglos A y B, flotantes de tamaño n y entregue un arreglo C que es la multiplicación de ambos componente a componente. Imprima C.
7. Cree un programa que reciba como entrada dos matrices A y B, flotantes de tamaño $n \times n$ y entregue un arreglo C que es la suma de ambos componente a componente. Imprima C.

Suma de prefijos
Introducción

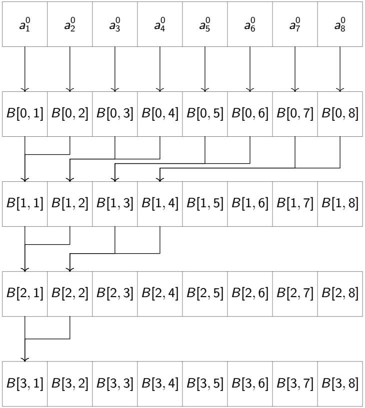
Consideremos el siguiente problema:
Suma de prefijos (Prefix-sum): Dado un arreglo $[a_1, \dots, a_n] \in A^n$ y $*$ una operación binaria interna asociativa en A. ¿Cuánto vale $a_1 * a_2 * \dots * a_n$?

SumaDePrefijos(a, n)

Entrada: $a = [a_1, \dots, a_n] \in A^n, \; n = 2^l$

Salida: $a_1 * \dots * a_n$

```
Para i = 1...n hSDFSDFacer en paralelo
  | B[0, i] = a[i]
Para t = 1....l hacer
  | Para i = 1....2^(l−t) hSDFSDFacer en paralelo
  |   | B[t, i] = B[t−1, 2i−1] ∗ B[t−1, 2i]
Devolver B[l, 1]
```



Suma de prefijos

Implementar el algoritmo de suma de prefijos

Ejercicio

1. Haga un pseudocódigo de un algoritmo que calcule la suma de prefijos utilizando,en lugar de la matriz B de dimensiones $n \times n$, un vector de tamaño $2n$ que guarde la suma actual y anterior.
2. Implemente el algoritmo de suma de prefijos en Pycuda usando memoria compartida.
3. Grafique los tiempos de ejecución en un plano tamaño de la entrada vs tiempo.

SumaDePrefijos(a, n) (doble búfer: a_{old}, a_{new})

Entrada: $a = [a_1, \ldots, a_n] \in A^n$, $n = 2^l$

Salida: $a_1 * \cdots * a_n$

Para $i = 1 \ldots n$ **hSDFSDFacer en paralelo**
- $a_{\text{old}}[i] \leftarrow a[i]$

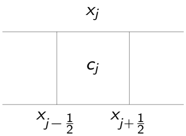
Para $t = 1 \ldots l$ **hacer**
- **Para** $i = 1 \ldots 2^{l-t}$ **hSDFSDFacer en paralelo**
  - $a_{\text{new}}[i] \leftarrow a_{\text{old}}[2i-1] * a_{\text{old}}[2i]$
- swap(a_{old}, a_{new}) ; `// intercambia punteros (sin copiar)`

Devolver $a_{\text{old}}[1]$

$$\text{Ec. de Burgers}: \begin{cases} u_t + (f(u))_x = 0, \quad x \in \mathbb{R}, \; t > 0, \\ f(u) = \frac{1}{2}u^2 \\ +\text{C.I.} + \text{C.C.} \end{cases}$$

Ecuaciones Diferenciales Parciales:

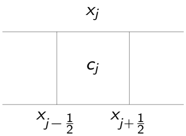
Esquema de Volúmenes Finitos



$$u_t = -(f(u))_x \qquad\qquad / \int_{c_j} \cdot dx$$

Ecuaciones Diferenciales Parciales:

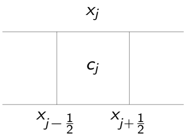
Esquema de Volúmenes Finitos



$$u_t = -(f(u))_x \qquad / \int_{c_j} \cdot dx$$

$$\frac{d}{dt}\int_{c_j} u(t,x)dx = -f\left(u(x_{j+\frac{1}{2}},t)\right) + f\left(u(x_{j-\frac{1}{2}},t)\right) \qquad / \int_{t_n}^{t_{n+1}} \cdot dt$$

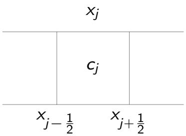
Ecuaciones Diferenciales Parciales:
Esquema de Volúmenes Finitos



$$u_t = -(f(u))_x \qquad / \int_{c_j} \cdot dx$$

$$\frac{d}{dt}\int_{c_j} u(t,x)dx = -f\left(u(x_{j+\frac{1}{2}},t)\right) + f\left(u(x_{j-\frac{1}{2}},t)\right) \qquad / \int_{t_n}^{t_{n+1}} \cdot dt$$

$$\int_{c_j} u(t^{n+1},x)dx - \int_{c_j} u(t^n,x)dx = -\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt + \int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt \qquad / \frac{1}{\Delta x}$$

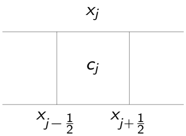


$$u_t = -(f(u))_x \qquad / \int_{c_j} \cdot dx$$

$$\frac{d}{dt}\int_{c_j} u(t,x)dx = -f\left(u(x_{j+\frac{1}{2}},t)\right) + f\left(u(x_{j-\frac{1}{2}},t)\right) \qquad / \int_{t_n}^{t_{n+1}} \cdot dt$$

$$\int_{c_j} u(t^{n+1},x)dx - \int_{c_j} u(t^n,x)dx = -\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right) dt + \int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right) dt \qquad / \frac{1}{\Delta x}$$

$$\frac{\int_{c_j} u(t^{n+1},x)dx}{\Delta x} - \frac{\int_{c_j} u(t^n,x)dx}{\Delta x} = -\frac{1}{\Delta x}\left[\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right) dt - \int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right) dt\right]$$



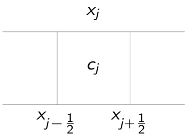
$$u_t = -(f(u))_x \qquad / \int_{c_j} \cdot dx$$

$$\frac{d}{dt}\int_{c_j} u(t,x)dx = -f\left(u(x_{j+\frac{1}{2}},t)\right) + f\left(u(x_{j-\frac{1}{2}},t)\right) \qquad / \int_{t_n}^{t_{n+1}} \cdot dt$$

$$\int_{c_j} u(t^{n+1},x)dx - \int_{c_j} u(t^n,x)dx = -\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt + \int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt \qquad / \frac{1}{\Delta x}$$

$$\frac{\int_{c_j} u(t^{n+1},x)dx}{\Delta x} - \frac{\int_{c_j} u(t^n,x)dx}{\Delta x} = -\frac{1}{\Delta x}\left[\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt - \int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt\right]$$

$$u_j^{n+1} - u_j^n = -\frac{\Delta t}{\Delta x}\left[\frac{\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt}{\Delta t} - \frac{\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt}{\Delta t}\right]$$



$$u_t = -(f(u))_x \qquad \Big/ \int_{c_j} \cdot\, dx$$

$$\frac{d}{dt}\int_{c_j} u(t,x)dx = -f\left(u(x_{j+\frac{1}{2}},t)\right) + f\left(u(x_{j-\frac{1}{2}},t)\right) \qquad \Big/ \int_{t_n}^{t_{n+1}} \cdot\, dt$$

$$\int_{c_j} u(t^{n+1},x)dx - \int_{c_j} u(t^n,x)dx = -\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt + \int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt \qquad \Big/ \frac{1}{\Delta x}$$

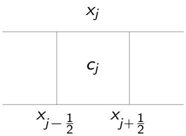
$$\frac{\int_{c_j} u(t^{n+1},x)dx}{\Delta x} - \frac{\int_{c_j} u(t^n,x)dx}{\Delta x} = -\frac{1}{\Delta x}\left[\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt - \int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt\right]$$

$$u_j^{n+1} - u_j^n = -\frac{\Delta t}{\Delta x}\left[\frac{\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt}{\Delta t} - \frac{\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt}{\Delta t}\right]$$

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left[F(u_j^n, u_{j+1}^n) - F(u_{j-1}^n, u_j^n)\right]$$

Ecuaciones Diferenciales Parciales:

Esquema de Volúmenes Finitos



$$u_t = -(f(u))_x \qquad / \int_{c_j} \cdot dx$$

$$\frac{d}{dt}\int_{c_j} u(t,x)dx = -f\left(u(x_{j+\frac{1}{2}},t)\right) + f\left(u(x_{j-\frac{1}{2}},t)\right) \qquad / \int_{t_n}^{t_{n+1}} \cdot dt$$

$$\int_{c_j} u(t^{n+1},x)dx - \int_{c_j} u(t^n,x)dx = -\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt + \int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt \qquad / \frac{1}{\Delta x}$$

$$\frac{\int_{c_j} u(t^{n+1},x)dx}{\Delta x} - \frac{\int_{c_j} u(t^n,x)dx}{\Delta x} = -\frac{1}{\Delta x}\left[\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt - \int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt\right]$$

$$u_j^{n+1} - u_j^n = -\frac{\Delta t}{\Delta x}\left[\frac{\int_{t^n}^{t^{n+1}} f\left(u(x_{j+\frac{1}{2}},t)\right)dt}{\Delta t} - \frac{\int_{t^n}^{t^{n+1}} f\left(u(x_{j-\frac{1}{2}},t)\right)dt}{\Delta t}\right]$$

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left[F(u_j^n, u_{j+1}^n) - F(u_{j-1}^n, u_j^n)\right]$$

$$u_j^{n+1} = \left(u_{j-1}^n + u_{j+1}^n\right)\left(\frac{1}{2} - \frac{1}{4}\frac{\Delta t}{\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right)\right). \qquad \text{por Lax-Friedrichs}$$

La condición de Courant-Friedrichs-Lewy es una condición de convergencia de algunos métodos de Volúmenes Finitos.

$$\max_{\Omega}(|f'(u(t))|)\frac{\Delta t}{\Delta x} < C$$

$$\Delta t < \frac{\Delta x}{\max_{\Omega}(|f'(u(t))|)}C$$

Además tenemos la siguiente relación entre la cantidad de celdas espaciales (n_x) y la cantidad de pasos de tiempo (n_t):

$$\max_{\Omega}(|f'(u(t))|)\frac{\Delta t}{\Delta x} < C$$

$$\max_{\Omega}(|f'(u(t))|)\frac{T/n_t}{L/n_x} < C$$

$$\max_{\Omega}(|f'(u(t))|)\frac{Tn_x}{L} < Cn_t$$

$$n_x\frac{T\max_{\Omega}(|f'(u(t))|)}{CL} < n_t$$

Ecuaciones Diferenciales Parciales:
Algoritmo

En términos generales, el algoritmo tienen la siguiente estructura:

Entrada: Parámetros EDP (C.I.,C.C., cte) y met. num. (Δx, CFL, etc)

Salida: Solución de la EDP en un tiempo dado.

t=0;
Mientras $T_{Actual} < T$ **hacer**
  Para $i = 1, ..., n_x - 1$ **hacer**
    $$u_i^{t+1} = u_i^t + \frac{\Delta t}{\Delta x}\left(F(u_i^t, u_{i+1}^t) - F(u_{i-1}^t, u_i^t)\right);$$
  $u_0^{t+1}, u_{n_x}^{t+1} = \textit{Condición de borde}$;
  $T_{\text{Actual}} = T_{\text{Actual}} + \Delta t$;
Devolver u;

Ecuaciones Diferenciales Parciales:
Algoritmo

En términos generales, el algoritmo tienen la siguiente estructura:

Entrada: Parámetros EDP (C.I.,C.C., cte) y met. num. (Δx, CFL, etc)

Salida: Solución de la EDP en un tiempo dado.

t=0;
Mientras $T_{Actual} < T$ **hacer**
    Para $i = 1, ..., n_x - 1$ **hacer**
        $u_i^{t+1} = u_i^t + \dfrac{\Delta t}{\Delta x}\left(F(u_i^t, u_{i+1}^t) - F(u_{i-1}^t, u_i^t)\right);$
    $u_0^{t+1}, u_{n_x}^{t+1} = \textit{Condición de borde};$
    $T_{\text{Actual}} = T_{\text{Actual}} + \Delta t;$
Devolver u;

En su versión paralela, el algoritmo tiene la siguiente forma:

Entrada: Parámetros EDP (C.I.,C.C., cte) y met. num. (Δx, CFL, etc)

Salida: Solución de la EDP en un tiempo dado.

t=0;
Mientras $T_{Actual} < T$ **hacer**
    para $i = 1, ..., n_x - 1$ **hacer in paralelo**
        $u_i^{t+1} = u_i^t + \dfrac{\Delta t}{\Delta x}\left(F(u_i^t, u_{i+1}^t) - F(u_{i-1}^t, u_i^t)\right);$
    $u_0^{t+1}, u_{n_x}^{t+1} = \textit{Condición de borde};$
    $T_{\text{Actual}} = T_{\text{Actual}} + \Delta t;$
Devolver u;

Consideremos el caso cuando la condición CFL es una igualdad (Δt máximo) y constante en el tiempo.

Caso secuencial:

$$T(n) = \mathcal{O}(n_t n_x) = \mathcal{O}(n_x^2)$$

Nota

Al pasar al modelo de cómputo paralelo, se intercambia procesadores por tiempo.

Caso secuencial:

Entrada: Parámetros EDP (C.I.,C.C., cte) y met. num. (Δx, CFL, etc)

Salida: Solución de la EDP en un tiempo dado.

t=0;
Mientras $T_{Actual} < T$ **hacer**
  Para $i = 1, \ldots, n_x - 1$ **hacer**
    $u_i^{t+1} = u_i^t + \dfrac{\Delta t}{\Delta x}\left(F(u_i^t, u_{i+1}^t) - F(u_{i-1}^t, u_i^t)\right)$;
  $u_0^{t+1}, u_{n_x}^{t+1} = \textit{Condición de borde}$;
  $T_{\text{Actual}} = T_{\text{Actual}} + \Delta t$;
Devolver u;

Consideremos el caso cuando la condición CFL es una igualdad (Δt máximo) y constante en el tiempo.

Caso secuencial:

$$T(n) = \mathcal{O}(n_t n_x) = \mathcal{O}(n_x^2)$$

Caso paralelo:

$$T(n) = \mathcal{O}(n_t) = \mathcal{O}(n_x)$$
$$P(n) = \mathcal{O}(n_x)$$

Nota

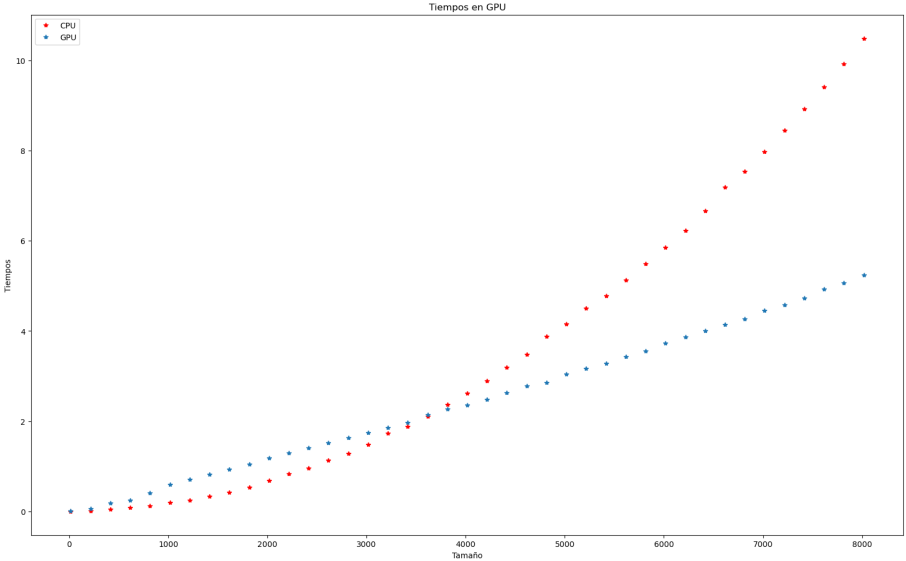
Al pasar al modelo de cómputo paralelo, se intercambia procesadores por tiempo.

Caso paralelo:

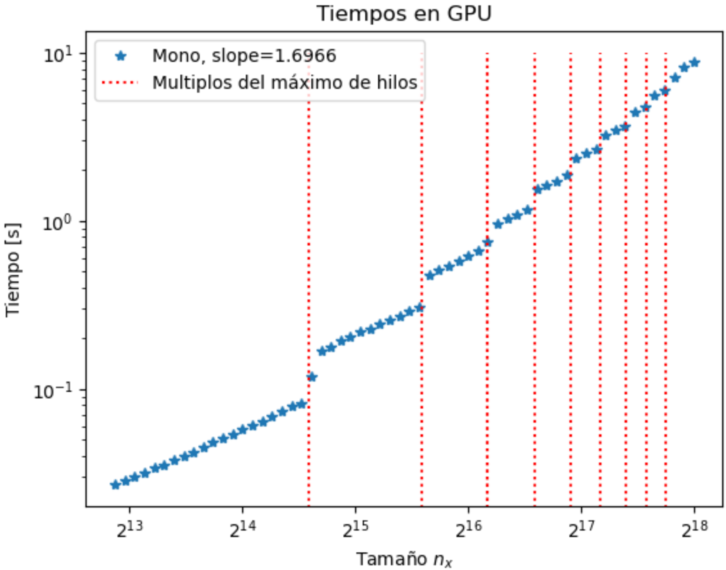
Entrada: Parámetros EDP (C.I.,C.C., cte) y met. num. (Δx, CFL, etc)

Salida: Solución de la EDP en un tiempo dado.

t=0;
Mientras $T_{Actual} < T$ **hacer**
  para $i = 1, \ldots, n_x - 1$ **hacer in paralelo**
    $u_i^{t+1} = u_i^t + \dfrac{\Delta t}{\Delta x}\left(F(u_i^t, u_{i+1}^t) - F(u_{i-1}^t, u_i^t)\right)$;
  $u_0^{t+1}, u_{n_x}^{t+1} = \mathit{Condición\ de\ borde}$;
  $T_{\text{Actual}} = T_{\text{Actual}} + \Delta t$;
Devolver u;



Tiempos en GPU
CPU
GPU
Tiempos
Tamaño
0
2
4
6
8
10
0
1000
2000
3000
4000
5000
6000
7000
8000


Tiempos en GPU
Mono, slope=1.6966
Multiplos del máximo de hilos
Tiempo [s]
10^1
10^0
10^{-1}
2^{13}
2^{14}
2^{15}
2^{16}
2^{17}
2^{18}
Tamaño n_x