

# Práctica 09. Cálculo de frecuencias



U.A.Q. Fac. de Informática  
**Dra. Sandra Luz Canchola Magdaleno**

Correo: [sandra.canchola@uaq.mx](mailto:sandra.canchola@uaq.mx)

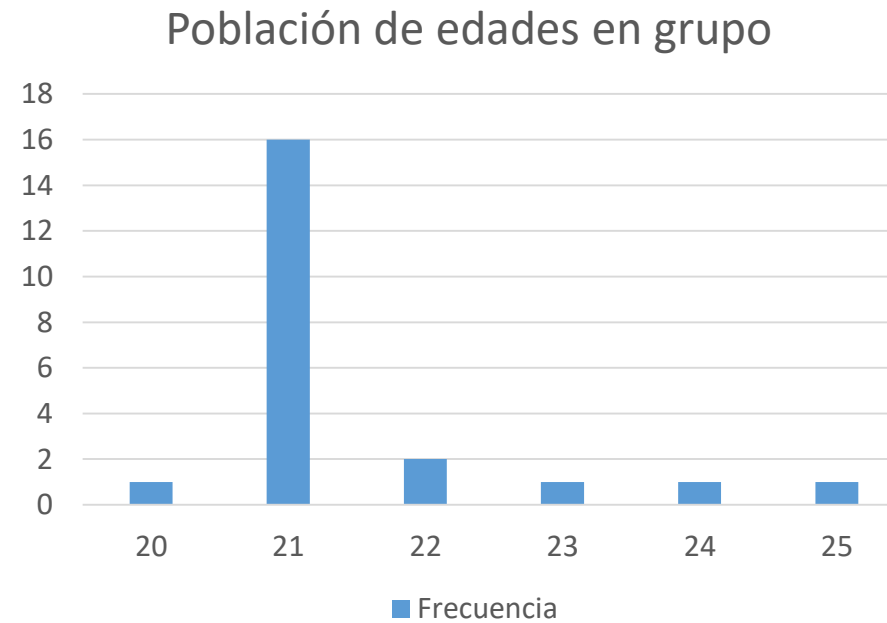
**Dra. Reyna Moreno Beltrán**

Correo: [reyna.moreno@uaq.mx](mailto:reyna.moreno@uaq.mx)

Cálculo de frecuencias. Es un estudio estadístico con el cual se define el número de veces (frecuencia) que un valor o un intervalo aparecen en una población muestra.

Población: 20, 24, 21, 21, 21, 21, 21, 21, 21, 23, 22, 22, 21, 21, 21, 25, 21, 21, 21, 21, 21, 21.

| Edad | Frecuencia |
|------|------------|
| 20   | 1          |
| 21   | 16         |
| 22   | 2          |
| 23   | 1          |
| 24   | 1          |
| 25   | 1          |



# Cálculo de frecuencia de valores en vectores n-Dimensionales

a

|       |       |       |     |           |
|-------|-------|-------|-----|-----------|
| 0     | 1     | 2     | ... | $n-1$     |
| $x_0$ | $x_1$ | $x_2$ |     | $x_{n-1}$ |

Vector de frecuencias

|       |       |       |     |           |
|-------|-------|-------|-----|-----------|
| $x_0$ | $x_1$ | $x_2$ | ... | $x_{n-1}$ |
| 1     | 1     | 1     |     | 1         |

# Problema de conflicto al leer la misma posición de un vector y modificarlo

datos

|   |   |    |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |
|---|---|----|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2  | 3 | 4 | 5 | 6 | 7 | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 8 | 9 | 10 | 8 | 7 | 6 | 8 | 7 | 10 | 10 | 9  | 8  | 6  | 7  | 10 | 8  | 9  | 7  | 6  | 8  |

Vector de frecuencias

|   |   |   |   |    |
|---|---|---|---|----|
| 6 | 7 | 8 | 9 | 10 |
| 3 | 4 | 4 | 3 | 3  |

$$\sum = 17 \neq n$$

# Operaciones atómicas

<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions>

Son operaciones que consideran el manejo con conflictos para el uso exclusivo de variables que son modificadas. Ejemplo:

- `atomicAdd()`
- `atomicSub()`
- `atomicExch()`
- `atomicMin()`
- `atomicMax()`
- `atomicDec()`
- `atomicCAS()`

# Proyecto CUDA

---



## Create a new project

Choose a project template with code scaffolding to get started



### CUDA 11.7 Runtime

A project that uses the CUDA 11.7 runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning



### CUDA 12.1 Runtime

A project that uses the CUDA 12.1 runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning

# Proyecto CUDA

---

## Configure your new project

CUDA 12.1 Runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning

Project name

Prog09\_CalculodeFrecuencias

Location

C:\TrabajoLaboratorio\CUDATopico2\Projects\Verano2024

...

Solution name ⓘ

Prog09\_CalculodeFrecuencias

☐

Place solution and project in the same directory

Project will be created in "C:\TrabajoLaboratorio\CUDATopico2\Projects\Verano2024  
\Prog09\_CalculodeFrecuencias\Prog09\_CalculodeFrecuencias\"

Back

Create

# Operaciones de memoria (CPU)

- **malloc.**- Reserva un bloque de memoria de un tamaño definido de bytes, retornando un apuntador al inicio de dicho bloque. El contenido de dicho bloque no se inicializa por lo que es indeterminado. Ejemplo:

```
void* malloc (size_t size);  
buffer = (char*) malloc (sizeof(char)*100);
```

- **memset.**- asigna valores en secciones de memoria. Ejemplo:  
Memset(variable, valor\_a\_asignar, tamaño\_de\_memoria)  
Donde: tamaño\_de\_memoria se define como n \* sizeof(tipo)

- **memcpy.**- Copia el contenido de un bloque de memoria referenciado por un apuntador a otro apuntador. Ejemplo:

```
void* memcpy( void* dest, const void* src, std::size_t count );  
memcpy (ptrDest, ptrOrigen, sizeof(int)*100);
```

- **free.**- Liberar la memoria reservada con el comando malloc. Ejemplo:  
free(pointerName);  
free(array2);



# Operaciones de memoria (GPU)

- **cudaMalloc**.- asigna una sección de memoria en GPU de acuerdo con el espacio solicitado.

Ejemplo:

```
cudaMalloc((void**) &apuntador, tamaño_de_memoria)
```

Donde: tamaño\_de\_memoria se define como  $n * \text{sizeof}(\text{tipo})$

- **cudaMemset**.- asigna valores en secciones de memoria.

Ejemplo:

```
Memset(apuntador, valor_a_asignar, tamaño_de_memoria)
```

Donde: tamaño\_de\_memoria se define como  $n * \text{sizeof}(\text{tipo})$

- **cudaMemcpy**.- copia memoria hacia y desde el device.

Ejemplo:

```
cudaMemcpy(destino, origen, tamaño_de_memoria, indicador_flujo_de_inf)
```

Donde Indicador= cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice

- **cudaFree**.- libera la memoria reservada por un apuntador.

Ejemplo:

```
cudaFree (apuntador)
```

# Memoria

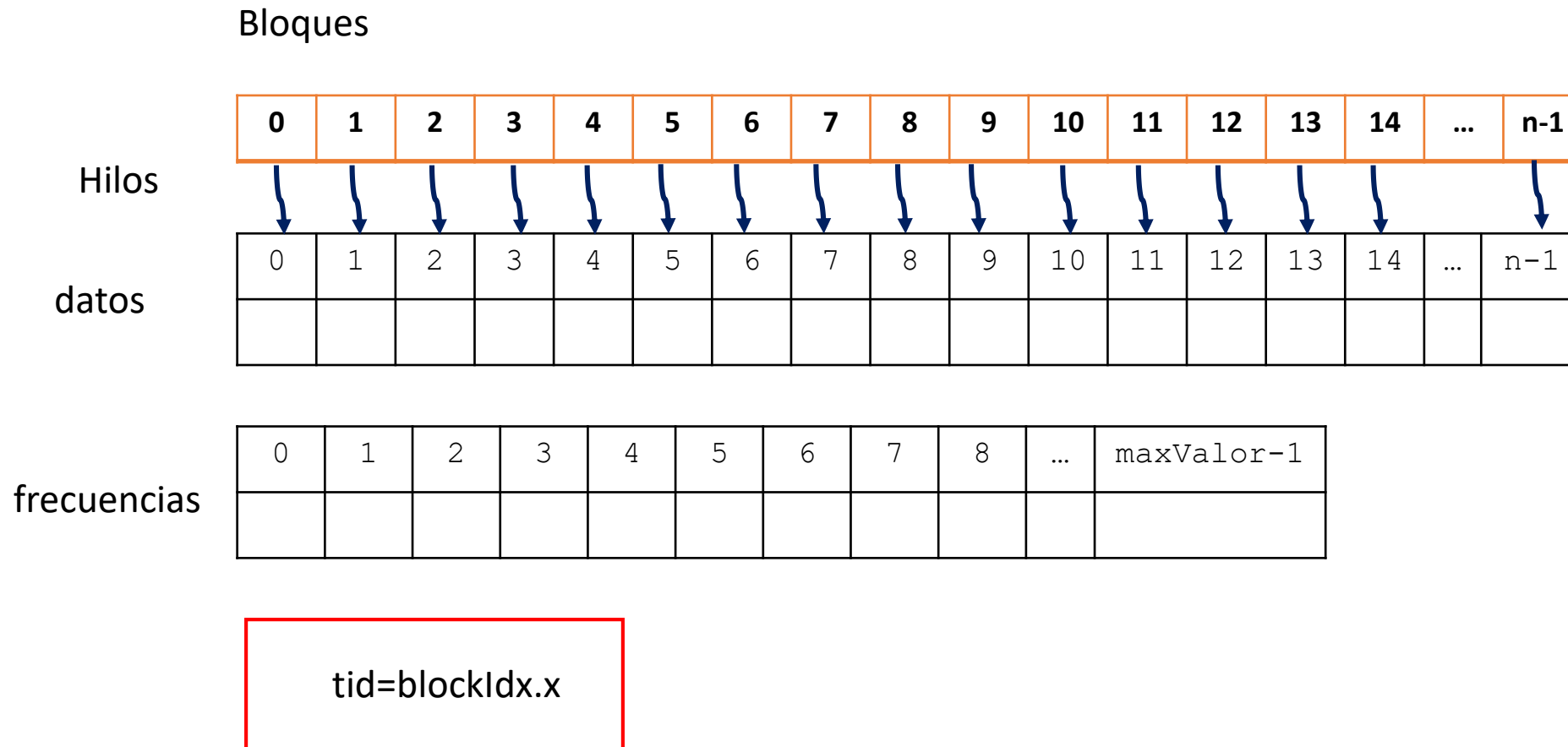
CPU (Host)

|     |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
|-----|-----------------|-----|---|----|---|----|---|----|---|----|---|----|---|----|---|----|---|-----|----|
| A01 | length          | 50  |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| A05 | hilosxBloque    | 5   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| A07 | maxValor        | 150 |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| A10 | datos           | α   | γ | φ  | φ | η  | χ | λ  | ε | τ  | β | κ  | θ | π  | δ | ε  | υ | ... | ω  |
| A15 | frecCPU         | f0  |   | f1 |   | f2 |   | f3 |   | f4 |   | f5 |   | f6 |   | f7 |   | ... | fn |
| A20 | frecuencias     | 0   |   | 0  |   | 0  |   | 0  |   | 0  |   | 0  |   | 0  |   | 0  |   | 0   | 0  |
| B01 | frecAtomic      | 0   |   | 0  |   | 0  |   | 0  |   | 0  |   | 0  |   | 0  |   | 0  |   | 0   | 0  |
| B45 | dev_datos       | J10 |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| B80 | dev_frecuencias | J45 |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| C30 | dev_frecAtomic  | J90 |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| E07 |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| E10 |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| F20 |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| G05 |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| H16 |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |
| H20 |                 |     |   |    |   |    |   |    |   |    |   |    |   |    |   |    |   |     |    |

GPU (Device)

|     |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
|-----|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|-----|---|
| J01 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| J05 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| J10 |  | α | γ | φ | φ | η | χ | λ | ε | τ | β | κ | θ | π | δ | ε | υ | ... | ω |
| J45 |  | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0   | 0 |
| J90 |  | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0 |   | 0   | 0 |
| K01 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| K05 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| K10 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| K15 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| K20 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| K30 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| L07 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |
| L10 |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |   |

# Caso 1. N bloques con hilo único



# Caso 1. N bloques con hilo único

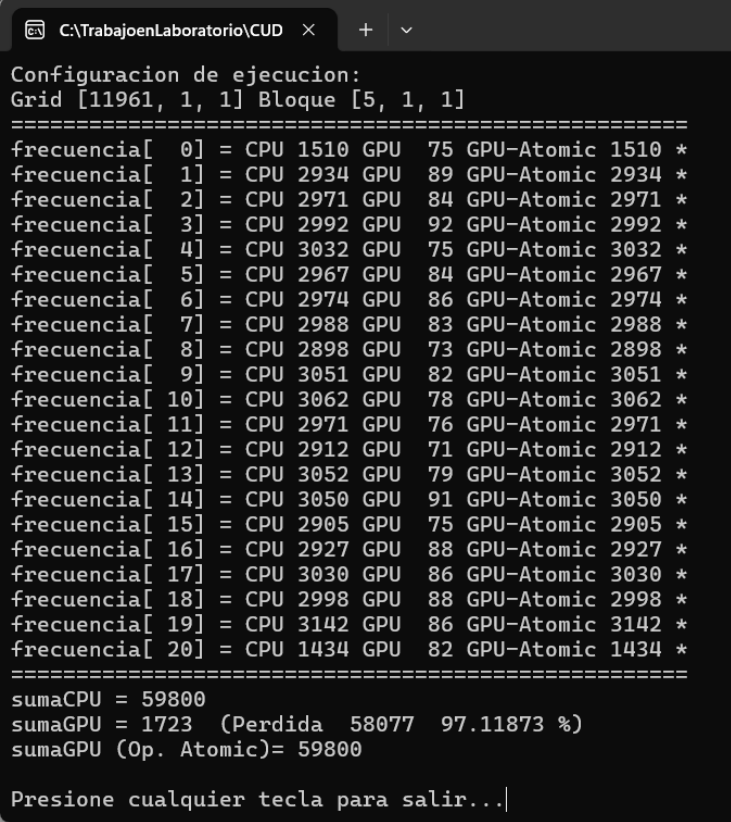
```
#define length 59800
#define maxValor 20
```

```
...
dim3 dimGrid(length);
dim3 dimBlock(1);
```

```
...
__global__ void calcularFrec(int* datos, int* frec) {
    int tid = blockIdx.x;
    int valor = datos[tid];
    frec[valor] = frec[valor] + 1;
}

__global__ void calcularFrecAtomic(int* datos, int* frec) {
    int tid = blockIdx.x;
    int valor = datos[tid];
    atomicAdd(&frec[valor], 1);
}
```

```
calcularFrec << <dimGrid, dimBlock >> > (dev_datos, dev_frecuencias);
cudaStatus = cudaGetLastError();
```

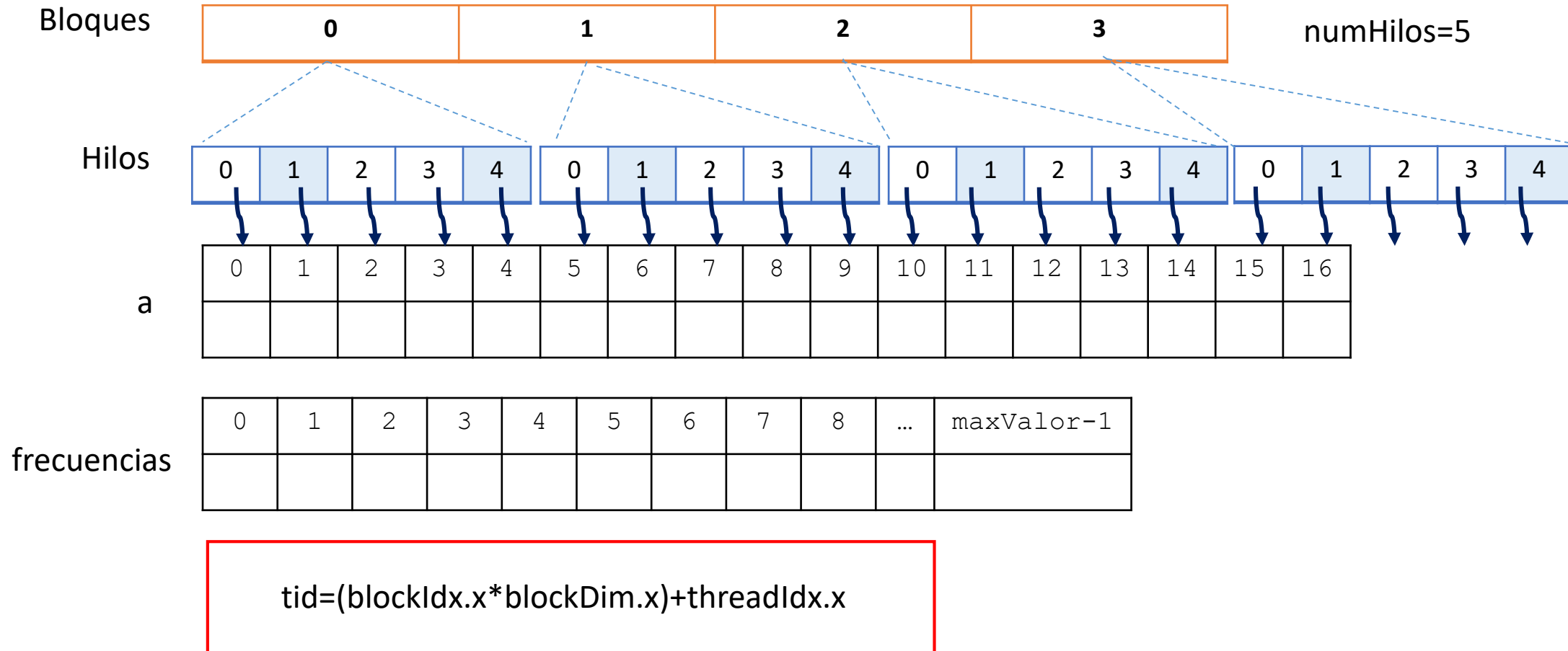


The screenshot shows a terminal window with the title "C:\TrabajoLaboratorio\CUD". It displays the output of a CUDA program. The output starts with "Configuracion de ejecucion:" followed by "Grid [11961, 1, 1] Bloque [5, 1, 1]". Below this is a table of frequencies for indices 0 to 20, comparing CPU and GPU results. The CPU results are consistent across all indices, while the GPU results show a significant discrepancy for indices 1 through 20. The summary at the bottom shows "sumaCPU = 59800", "sumaGPU = 1723 (Perdida 58077 97.11873 %)", and "sumaGPU (Op. Atomic)= 59800". The prompt "Presione cualquier tecla para salir..." is at the bottom.

```
C:\TrabajoLaboratorio\CUD >
Configuracion de ejecucion:
Grid [11961, 1, 1] Bloque [5, 1, 1]
=====
frecuencia[ 0] = CPU 1510 GPU 75 GPU-Atomic 1510 *
frecuencia[ 1] = CPU 2934 GPU 89 GPU-Atomic 2934 *
frecuencia[ 2] = CPU 2971 GPU 84 GPU-Atomic 2971 *
frecuencia[ 3] = CPU 2992 GPU 92 GPU-Atomic 2992 *
frecuencia[ 4] = CPU 3032 GPU 75 GPU-Atomic 3032 *
frecuencia[ 5] = CPU 2967 GPU 84 GPU-Atomic 2967 *
frecuencia[ 6] = CPU 2974 GPU 86 GPU-Atomic 2974 *
frecuencia[ 7] = CPU 2988 GPU 83 GPU-Atomic 2988 *
frecuencia[ 8] = CPU 2898 GPU 73 GPU-Atomic 2898 *
frecuencia[ 9] = CPU 3051 GPU 82 GPU-Atomic 3051 *
frecuencia[10] = CPU 3062 GPU 78 GPU-Atomic 3062 *
frecuencia[11] = CPU 2971 GPU 76 GPU-Atomic 2971 *
frecuencia[12] = CPU 2912 GPU 71 GPU-Atomic 2912 *
frecuencia[13] = CPU 3052 GPU 79 GPU-Atomic 3052 *
frecuencia[14] = CPU 3050 GPU 91 GPU-Atomic 3050 *
frecuencia[15] = CPU 2905 GPU 75 GPU-Atomic 2905 *
frecuencia[16] = CPU 2927 GPU 88 GPU-Atomic 2927 *
frecuencia[17] = CPU 3030 GPU 86 GPU-Atomic 3030 *
frecuencia[18] = CPU 2998 GPU 88 GPU-Atomic 2998 *
frecuencia[19] = CPU 3142 GPU 86 GPU-Atomic 3142 *
frecuencia[20] = CPU 1434 GPU 82 GPU-Atomic 1434 *
=====
sumaCPU = 59800
sumaGPU = 1723 (Perdida 58077 97.11873 %)
sumaGPU (Op. Atomic)= 59800

Presione cualquier tecla para salir...|
```

## Caso 2. X bloques con numHilos c/u



## Caso 2. X bloques con numHilos c/u

```
#define length 59800
#define maxValor 20
#define hilosxBloque 5
```

```
...
```

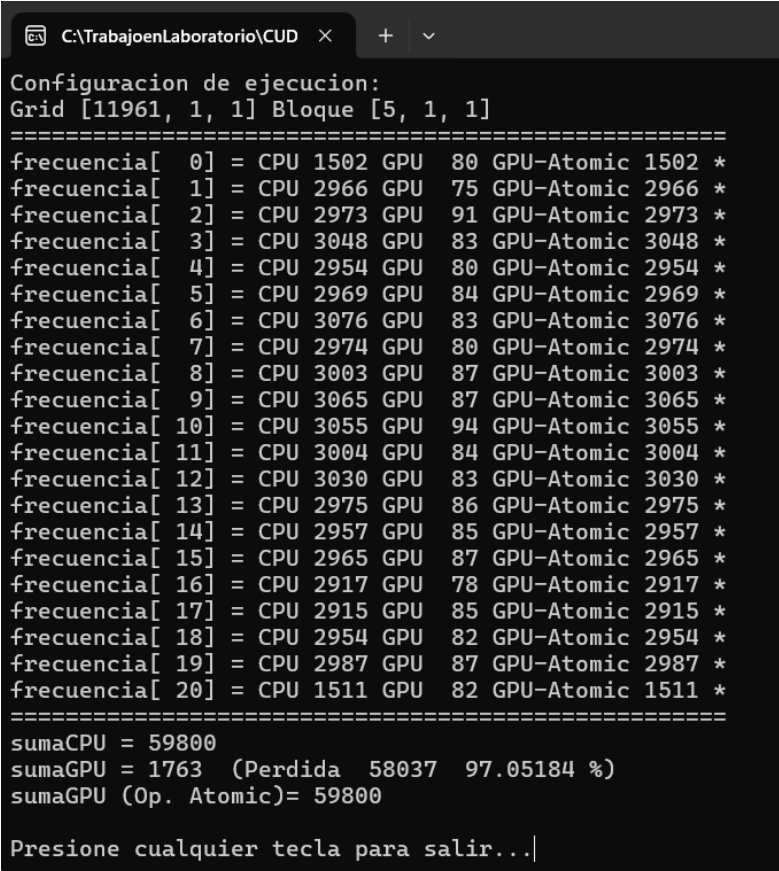
```
dim3 dimGrid((length / hilosxBloque) + 1);
dim3 dimBlock(hilosxBloque);
```

```
...
```

```
__global__ void calcularFrec(int* datos, int* frec) {
    int tid = (blockIdx.x * blockDim.x) + threadIdx.x;
    if (tid < length) {
        int valor = datos[tid];
        frec[valor] = frec[valor] + 1;}
}
```

```
__global__ void calcularFrecAtomic(int* datos, int* frec) {
    int tid = (blockIdx.x * blockDim.x) + threadIdx.x;
    if (tid < length) {
        int valor = datos[tid];
        atomicAdd(&frec[valor], 1);}
}
```

```
calcularFrec << <dimGrid, dimBlock >> > (dev_datos, dev_frecuencias);
cudaStatus = cudaGetLastError();
```



The screenshot shows a terminal window with the title "C:\TrabajoLaboratorio\CUD". It displays the output of a CUDA program. At the top, it shows the execution configuration: "Configuracion de ejecucion: Grid [11961, 1, 1] Bloque [5, 1, 1]". Below this is a table of results for 21 frequency values (0 to 20). Each row shows the frequency value, the CPU count, the GPU count, the GPU-Atomic count, and a star symbol. The results are as follows:

| frecuencia[ ] | CPU  | GPU | GPU-Atomic | * |
|---------------|------|-----|------------|---|
| 0             | 1502 | 80  | 1502       | * |
| 1             | 2966 | 75  | 2966       | * |
| 2             | 2973 | 91  | 2973       | * |
| 3             | 3048 | 83  | 3048       | * |
| 4             | 2954 | 80  | 2954       | * |
| 5             | 2969 | 84  | 2969       | * |
| 6             | 3076 | 83  | 3076       | * |
| 7             | 2974 | 80  | 2974       | * |
| 8             | 3003 | 87  | 3003       | * |
| 9             | 3065 | 87  | 3065       | * |
| 10            | 3055 | 94  | 3055       | * |
| 11            | 3004 | 84  | 3004       | * |
| 12            | 3030 | 83  | 3030       | * |
| 13            | 2975 | 86  | 2975       | * |
| 14            | 2957 | 85  | 2957       | * |
| 15            | 2965 | 87  | 2965       | * |
| 16            | 2917 | 78  | 2917       | * |
| 17            | 2915 | 85  | 2915       | * |
| 18            | 2954 | 82  | 2954       | * |
| 19            | 2987 | 87  | 2987       | * |
| 20            | 1511 | 82  | 1511       | * |

Below the table, it shows the sum of frequencies: "sumaCPU = 59800", "sumaGPU = 1763 (Perdida 58037 97.05184 %)", and "sumaGPU (Op. Atomic)= 59800". At the bottom, it says "Presione cualquier tecla para salir...".

# Bibliografía

---

- Documentación **CUDA C++ Programming Guide** NVIDIA. 2024  
<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- Sitio **CUDA Toolkit Documentation** NVIDIA, 2024.  
<https://docs.nvidia.com/cuda/index.html>
- Storti, Duane; Yurtoglu, Mete. **CUDA for Engineers:An Introduction to High-Performance Parallel Computing**. Addison Wesley. 2015.
- Cheng, John; Grossman, Max; McKercher. **Professional CUDA C Programming**. Edit. Wrox. 2014.
- Sanders, Jason; Kandrot, Edward. **CUDA by Example:An Introduction to General-Purpose GPU Programming**. Addison Wesley. 2011.
- Kirk, David; Hwu, Wen-mei. **Programming Massively Parallel Processors: A Hands-on Approach**. Elsevier. 2010.

---

Gracias por su atención



**U.A.Q. Fac. de Informática  
Campus Juriquilla**

**Dra. Sandra Luz Canchola Magdaleno**  
**sandra.canchola@uaq.mx**  
**Cel. 442-1369270**

**Dra. Reyna Moreno Beltrán**  
**reyna.moreno@uaq.mx**