Introducción a CUDA

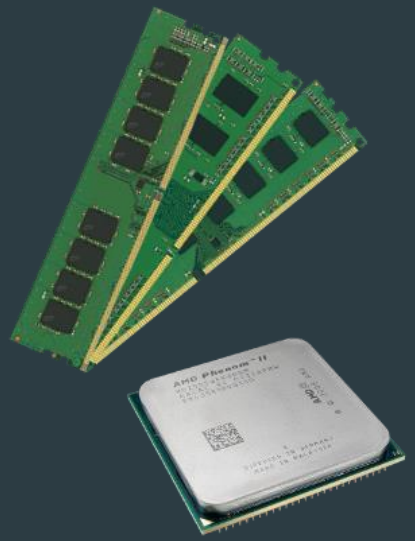
¿Qué es CUDA?

- Arquitectura CUDA
  - Permite hacer cómputo sobre GPU de propósito general

- CUDA C/C++
  - Extensión que permite el cómputo heterogéneo
  - API's para manejar los dispositivos y su memoria

Cómputo heterogéneo

- Terminología:
  - Host: El CPU y su memoria
  - Device: La GPU y su memoria



Host

Device

Cómputo heterogéneo

device code

host code

```
#include <iostream>
#include <algorithm>

using namespace std;

#define N          1024
#define RADIUS     3
#define BLOCK_SIZE 16

__global__ void stencil_1d(int *in, int *out) {
        __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
        int gindex = threadIdx.x + blockIdx.x * blockDim.x;
        int lindex = threadIdx.x + RADIUS;

        // Read input elements into shared memory
        temp[lindex] = in[gindex];
        if (threadIdx.x < RADIUS) {
                temp[lindex - RADIUS] = in[gindex - RADIUS];
                temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
        }

        // Synchronize (ensure all the data is available)
        __syncthreads();

        // Apply the stencil
        int result = 0;
        for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
                result += temp[lindex + offset];

        // Store the result
        out[gindex] = result;
}

void fill_ints(int *x, int n) {
        fill_n(x, n, 1);
}

int main(void) {
        int *in, *out;          // host copies of a, b, c
        int *d_in, *d_out;      // device copies of a, b, c
        int size = (N + 2*RADIUS) * sizeof(int);

        // Alloc space for host copies and setup values
        in = (int *)malloc(size); fill_ints(in,  N + 2*RADIUS);
        out = (int *)malloc(size); fill_ints(out, N + 2*RADIUS);

        // Alloc space for device copies
        cudaMalloc((void **)&d_in,  size);
        cudaMalloc((void **)&d_out, size);

        // Copy to device
        cudaMemcpy(d_in,  in,  size, cudaMemcpyHostToDevice);
        cudaMemcpy(d_out, out, size, cudaMemcpyHostToDevice);

        // Launch stencil_1d() kernel on GPU
        stencil_1d<<<N/BLOCK_SIZE,BLOCK_SIZE>>>(d_in + RADIUS, d_out + RADIUS);

        // Copy result back to host
        cudaMemcpy(out, d_out, size, cudaMemcpyDeviceToHost);

        // Cleanup
        free(in); free(out);
        cudaFree(d_in); cudaFree(d_out);
        return 0;
}
```

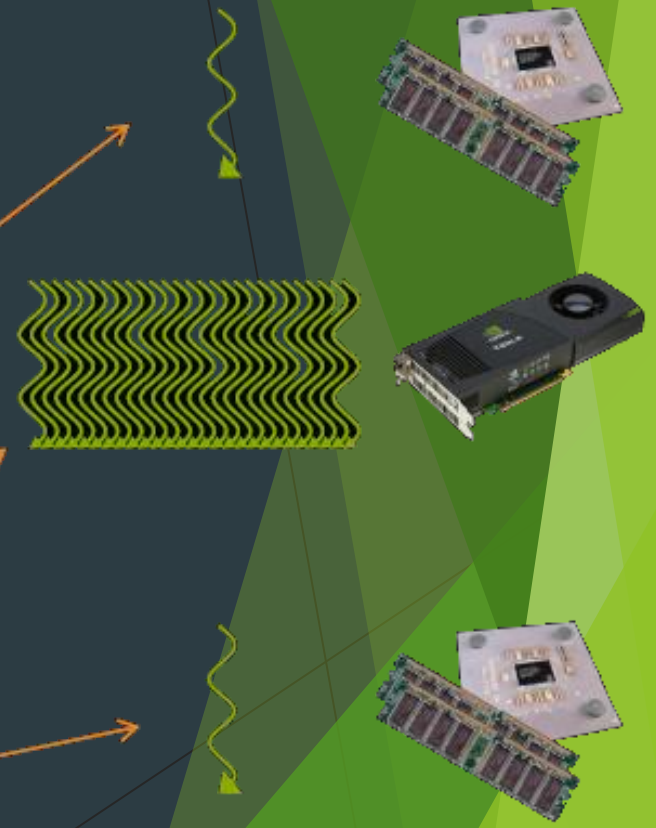
parallel function

serial function

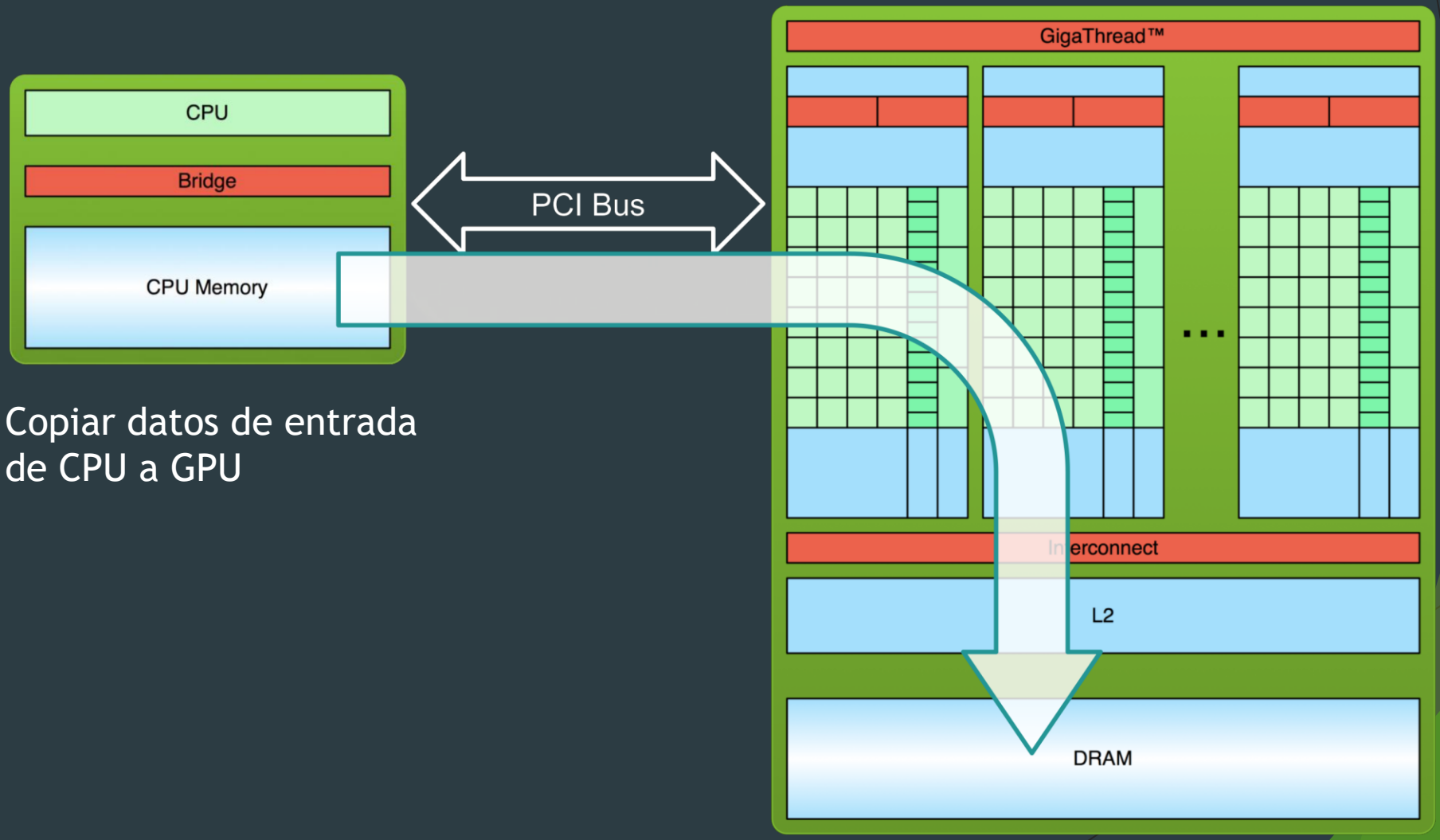
serial code

parallel code

serial code

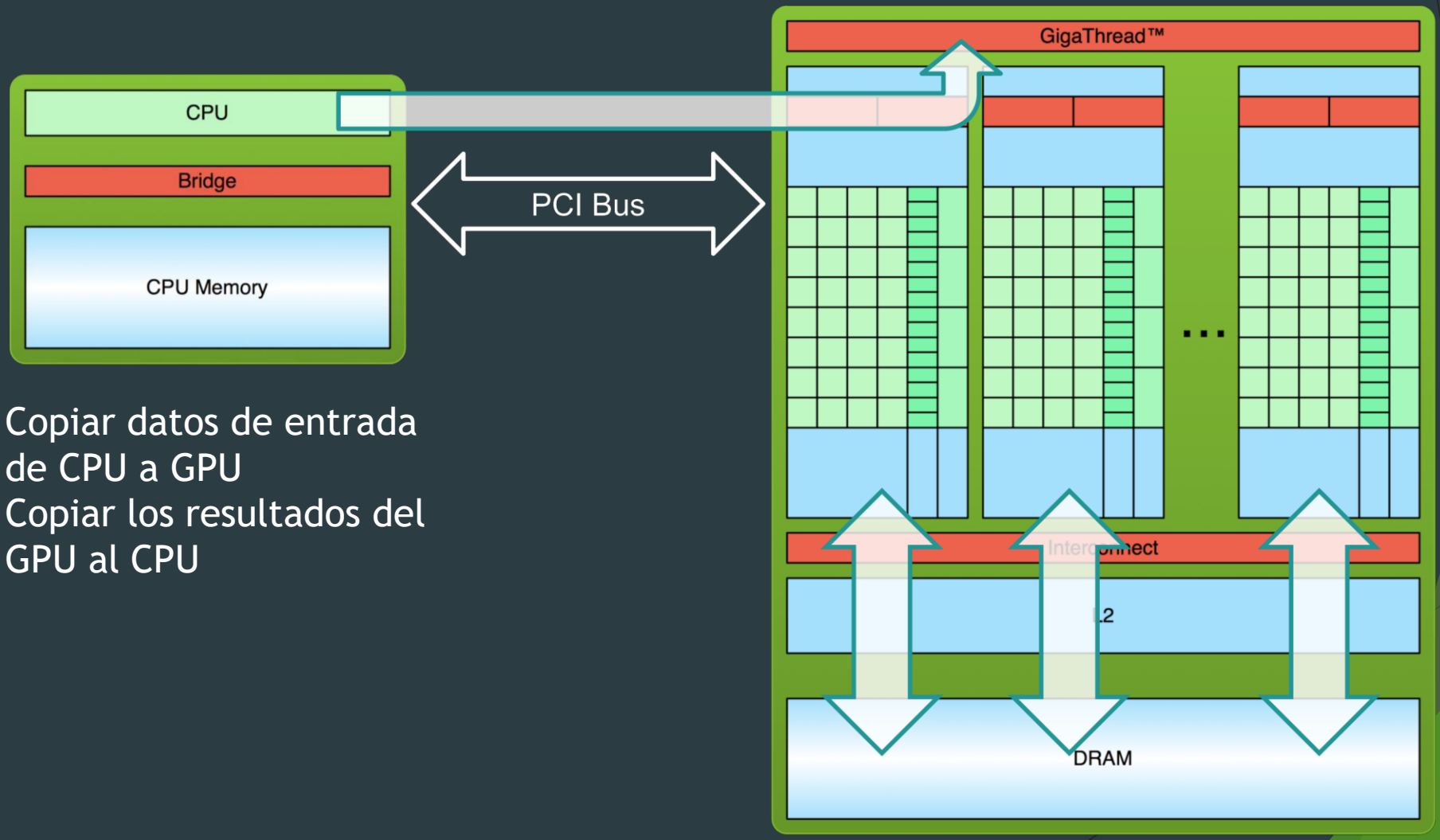


Cómputo heterogéneo



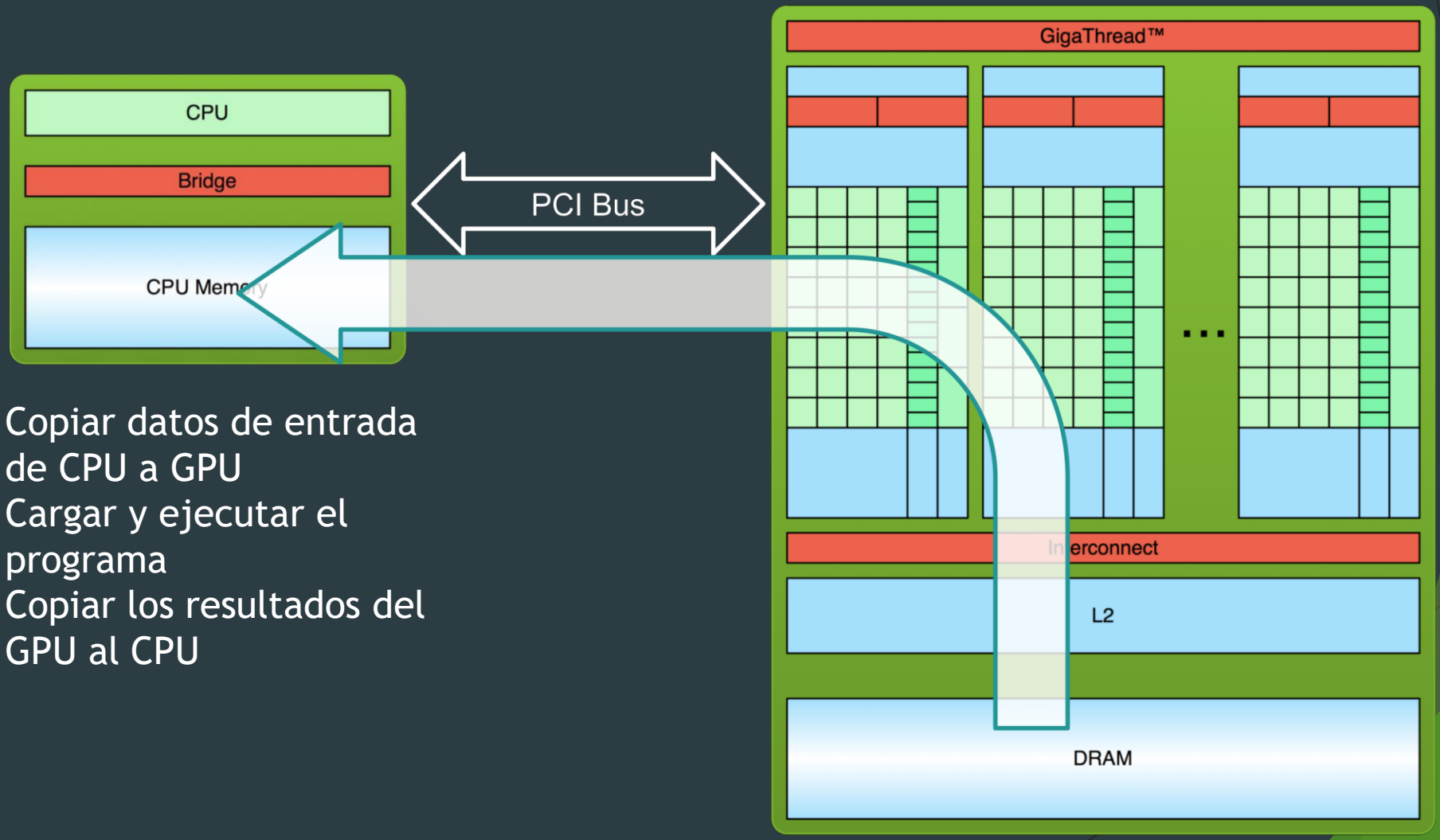
1. Copiar datos de entrada de CPU a GPU

Cómputo heterogéneo



1. Copiar datos de entrada de CPU a GPU
2. Copiar los resultados del GPU al CPU

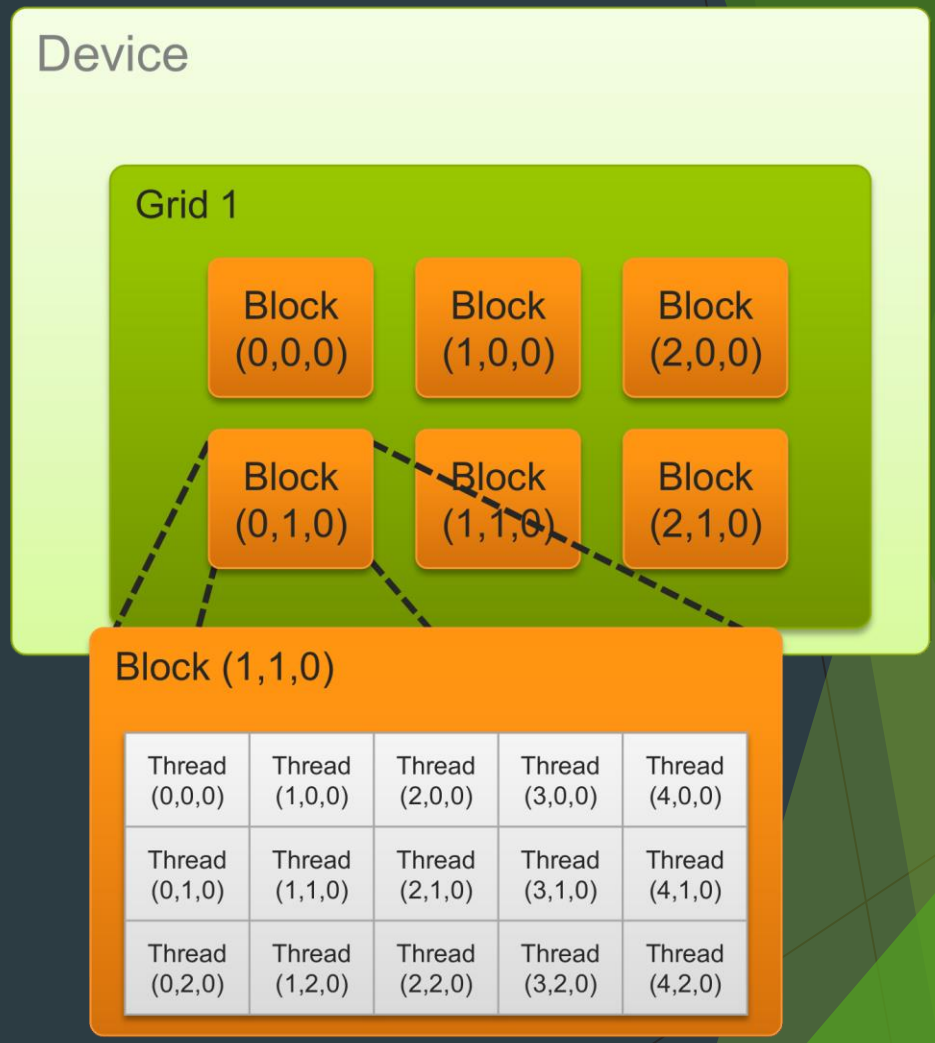
Cómputo heterogéneo



1. Copiar datos de entrada de CPU a GPU
2. Cargar y ejecutar el programa
3. Copiar los resultados del GPU al CPU

Arquitectura del dispositivo

- El código a ejecutarse sobre GPU es lanzado como una malla de bloques de hilos.
- Variables relevantes:
  - threadIdx
  - blockIdx
  - blockDim
  - gridDim



Hola mundo

```
int main() {
    printf("Hola mundo! \n");
    return 0;
}
```

- Programa sobre CPU (host)

- El compilador de NVIDIA (NVCC) puede ser utilizado para compilar códigos sin instrucciones sobre la GPU (device)

Salida:

```
$nvcc Hola_mundo.cu
$a.out
Hola mundo!
$
```

Hola mundo con código CUDA

```
__global__ void  mykernel( ){
}
int main() {
    mykernel<<<1,1>>>();
    printf("Hola mundo! \n");
    return 0;
}
```

- Dos elementos sintácticos nuevos

Hola mundo con código CUDA

```
__global__ void  mykernel( )
```

- La palabra reservada __global__ indica que la función:
  - Corre sobre la GPU
  - Es llamada desde código de CPU
- Nvcc separa el código de CPU y de GPU
  - mykernel( ) es procesado por el compilador de NVIDIA
  - main( ) es procesado por el compilador de C/C++

Hola mundo con código CUDA

```
mykernel<<<1,1>>>( );
```

- Los símbolos triples de desigualdad determinan una llamada de código de GPU desde el código de CPU
  - También conocido como “kernel launch”
  - (1,1) determinal la configuración de la malla

Hola mundo con código CUDA

```
__global__ void  mykernel( ){
}
int main() {
    mykernel<<<1,1>>>();
    printf("Hola mundo! \n");
    return 0;
}
```

- mykernel() hace nada

Salida:

```
$nvcc Hola_mundo.cu
$a.out
Hola mundo!
$
```