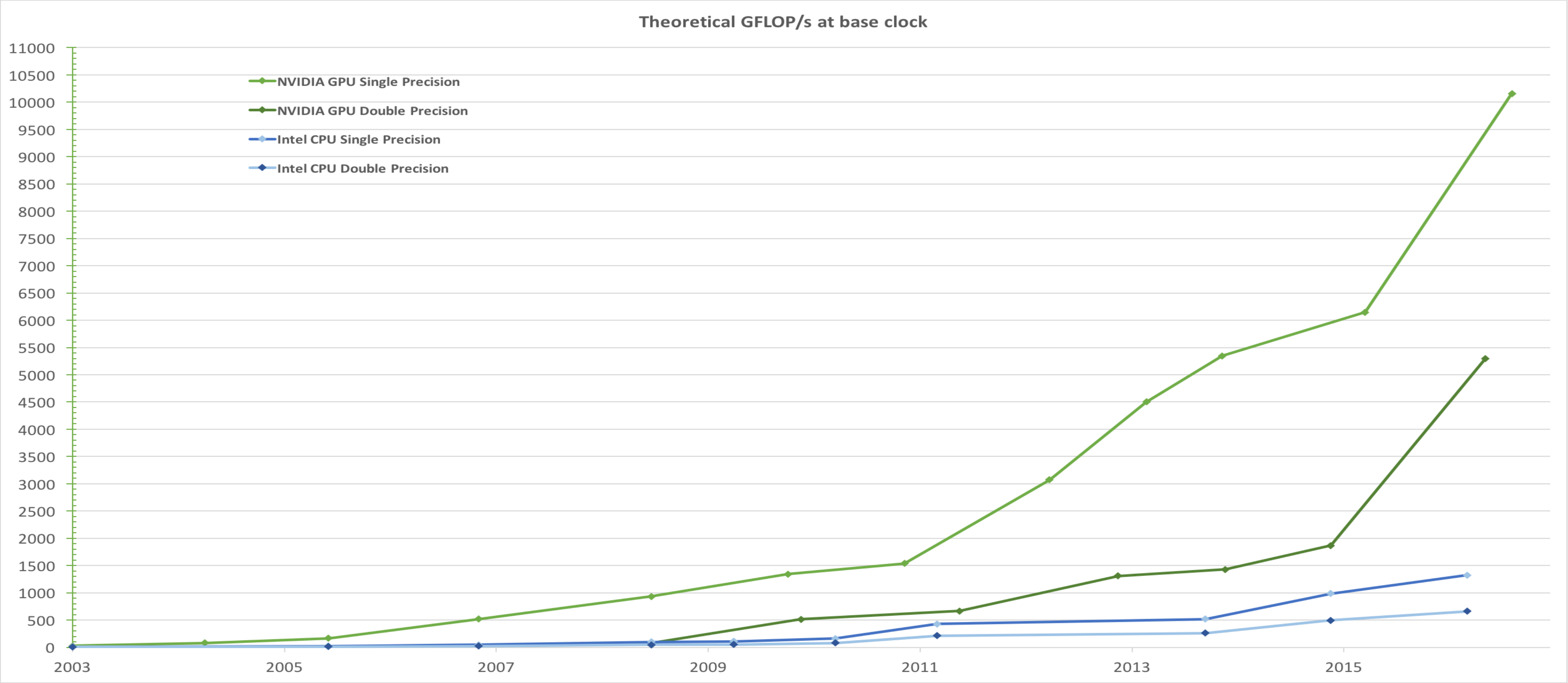
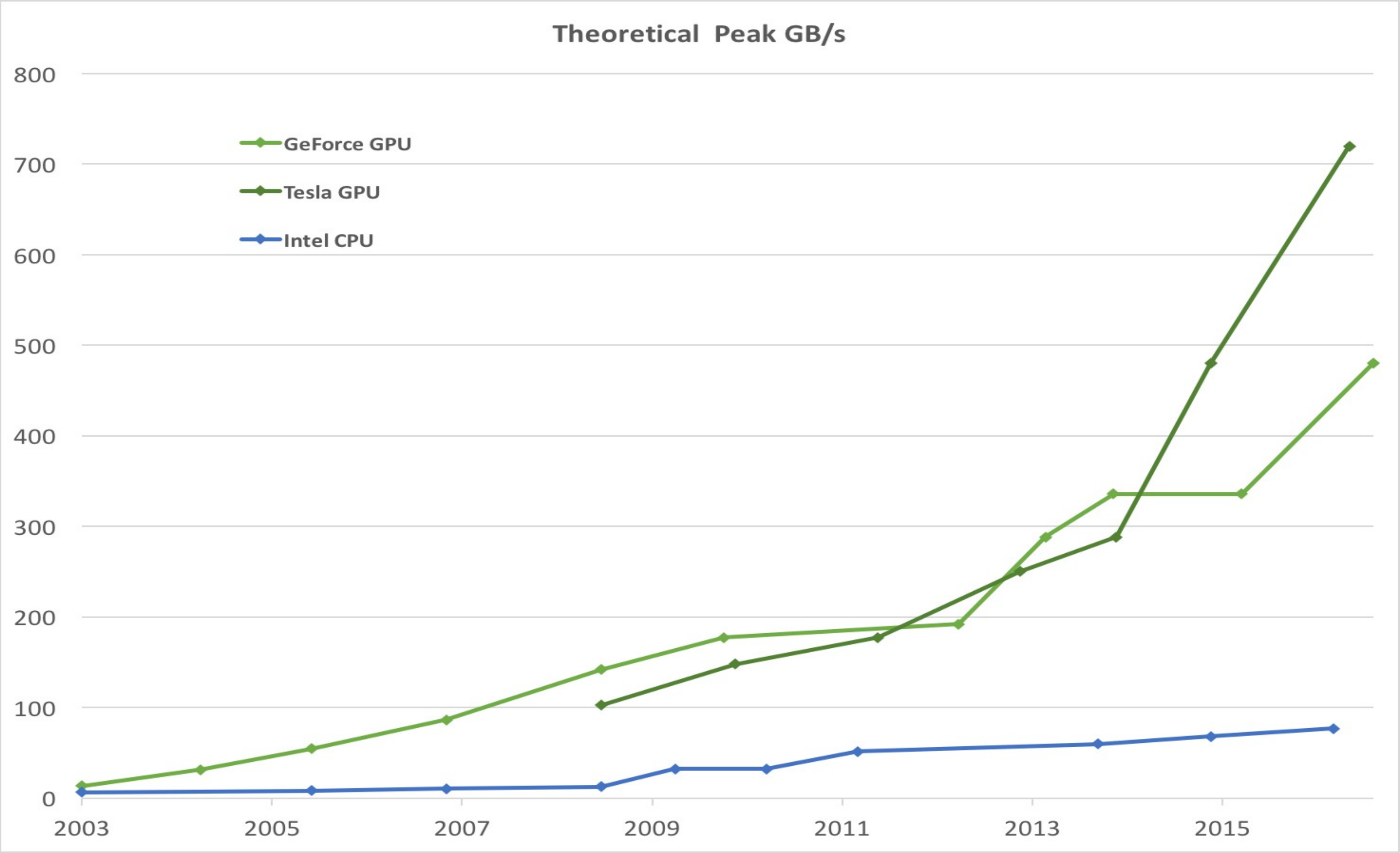
GPGPU Programming

Why GP-GPU?



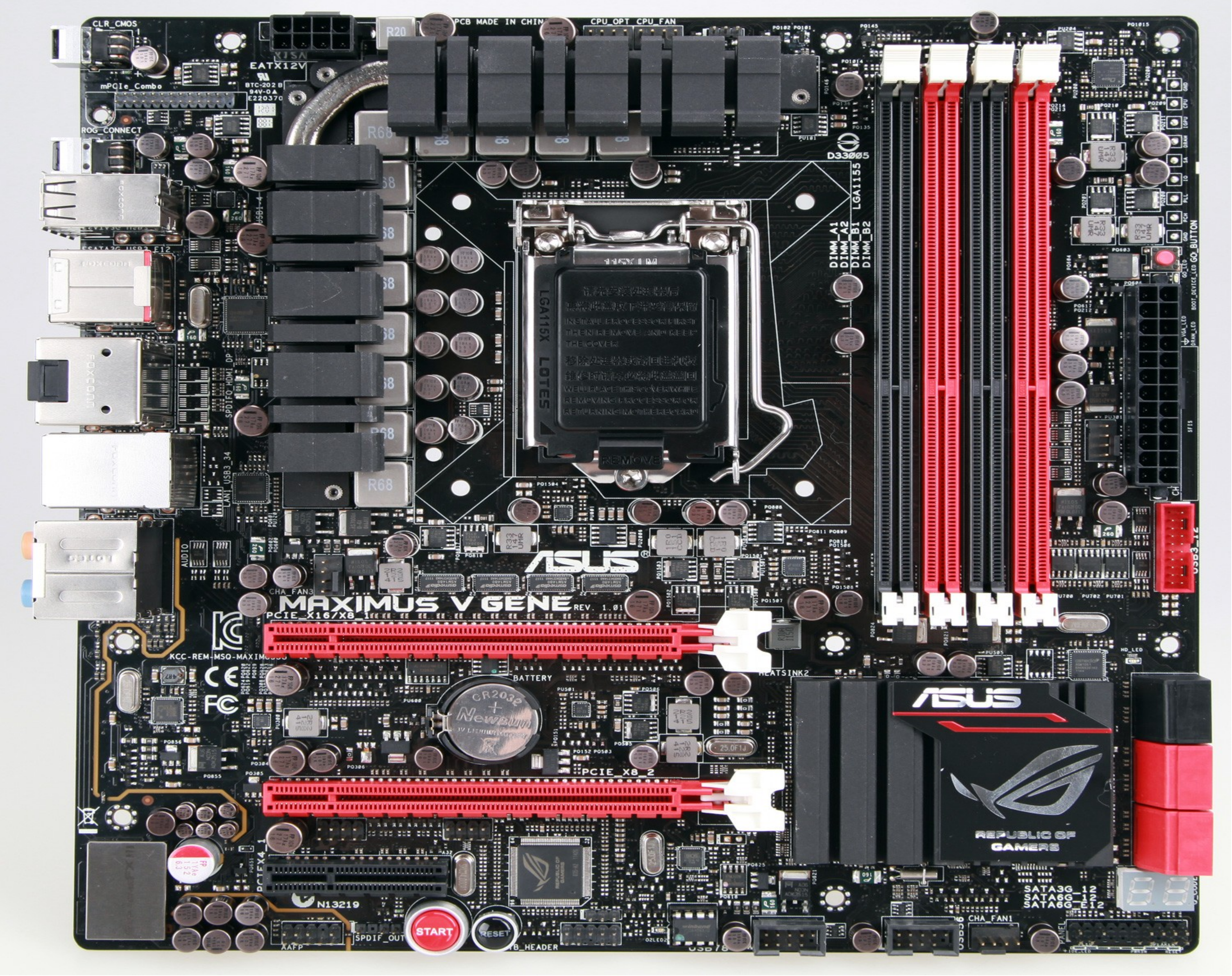
Source: CUDA C Programming Guide (NVIDIA)

Why GP-GPU?

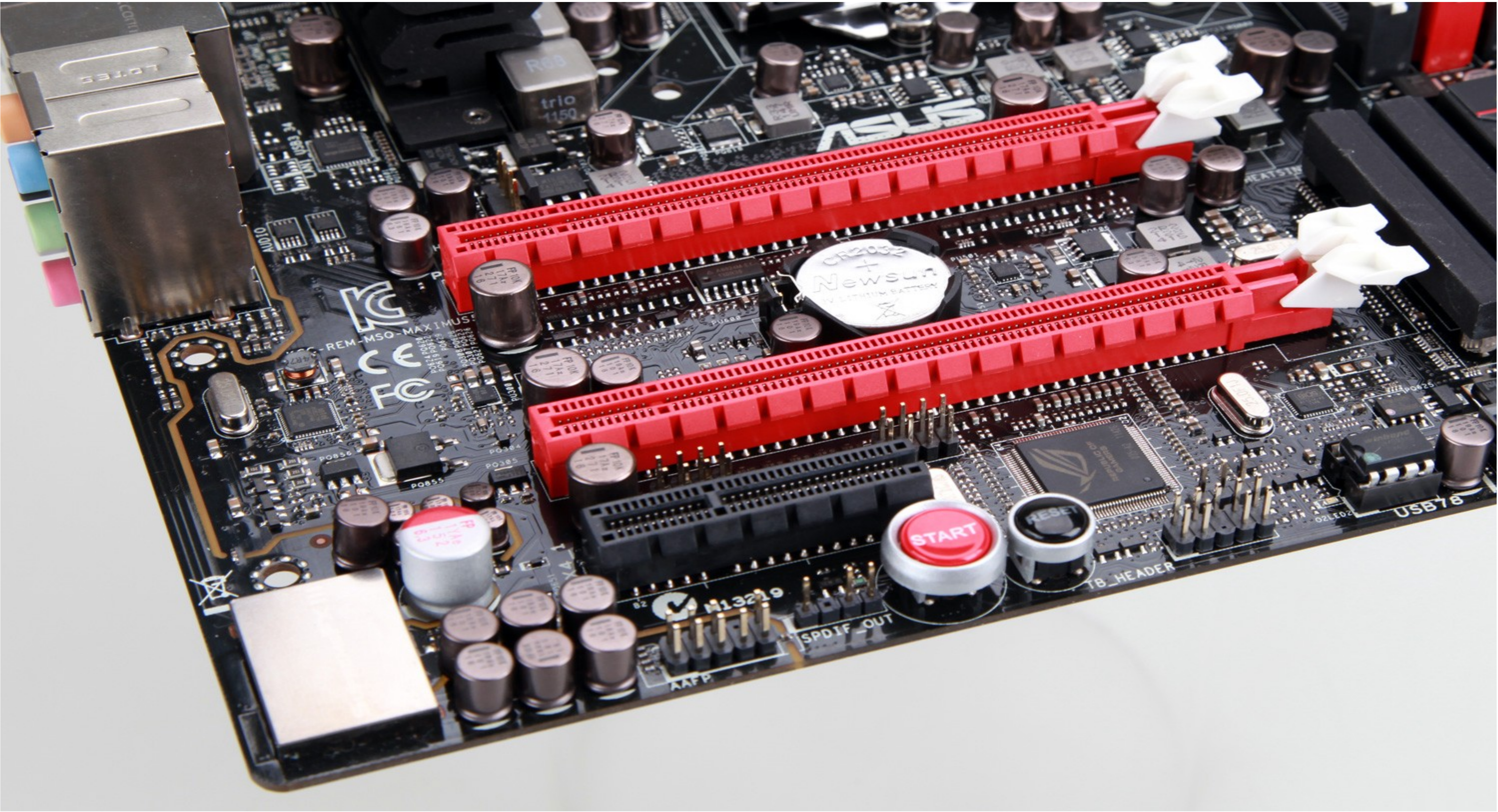


Source: CUDA C Programming Guide (NVIDIA)

Where is my GPU?



www.asus.com



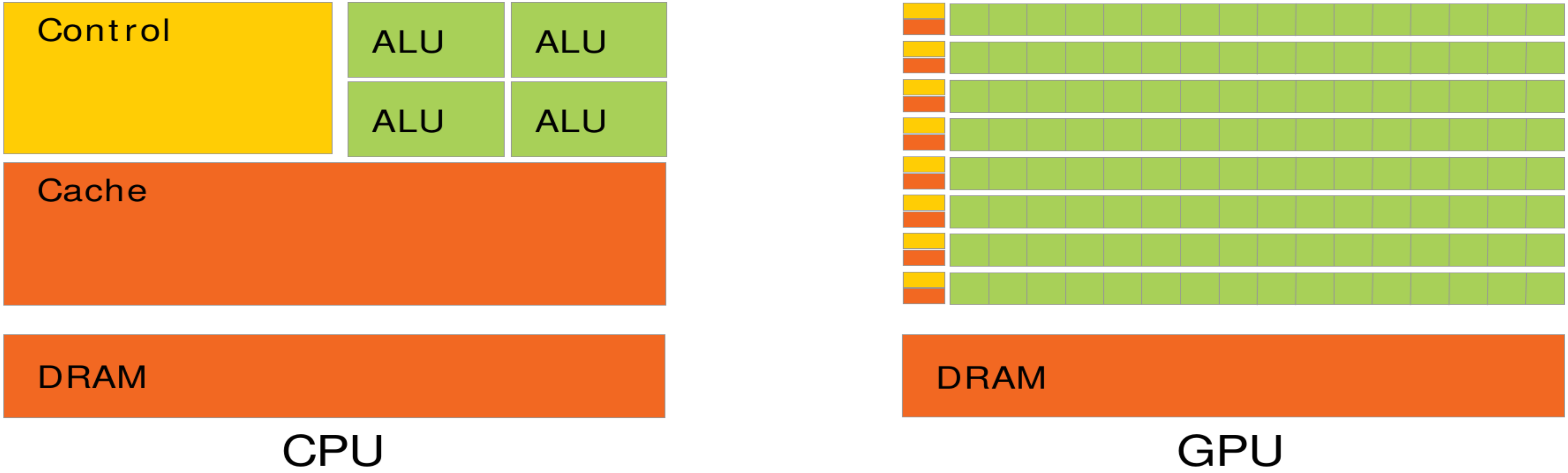
www.asus.com

www.asus.com

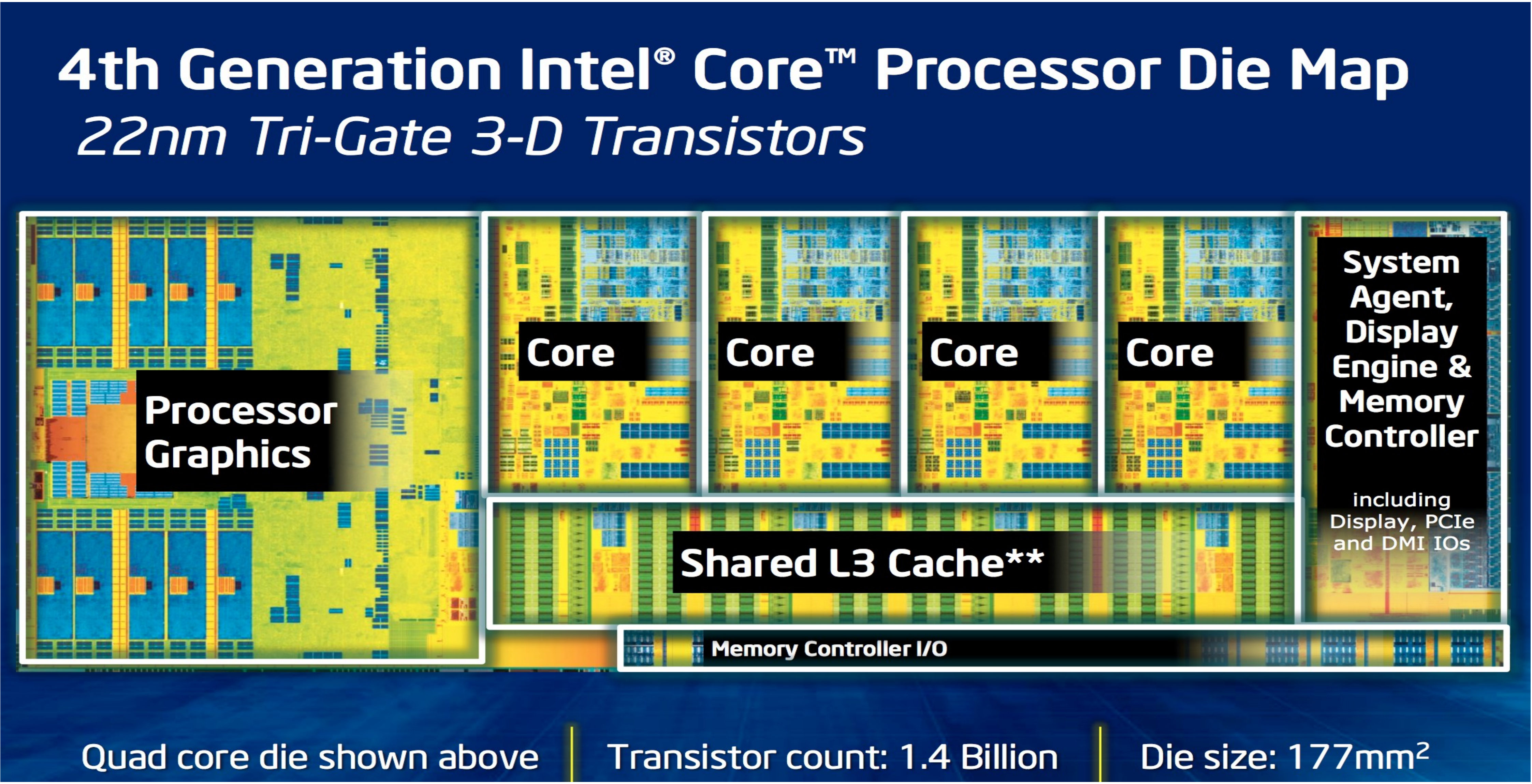
www.asus.com

How it differs from CPU?

CPU vs GPU



Source: CUDA C Programming Guide (NVIDIA)



4th Generation Intel® Core™ Processor Die Map

22nm Tri-Gate 3-D Transistors

Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O

Quad core die shown above | Transistor count: 1.4 Billion | Die size: 177mm²

http://images.anandtech.com/doci/7003/Screen%20Shot%202013-05-31%20at%207.59.16%20PM.png

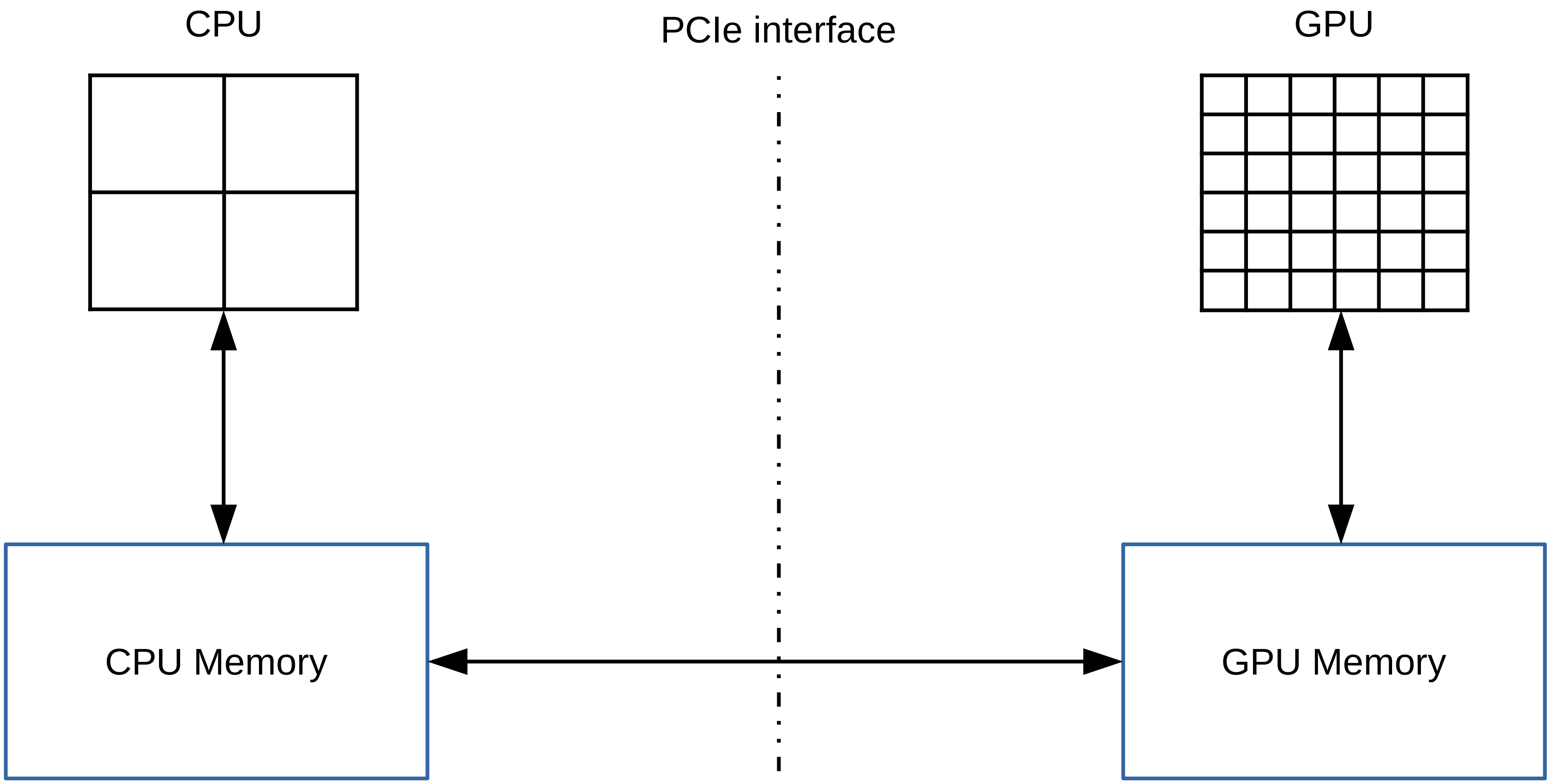
CPU vs GPU

- CPU
  - Optimized for serial thread performance
  - Good for complex tasks
  - Few, large, complex cores
  - Large number of transistors are allocated for Caches, Instruction Level Parallelism

- GPU
  - Optimized for data parallel, throughput computations
  - Large number of very small, simple cores
  - More number of transistors are allocated for computations

How to program these GPUs for general purpose computing?

Programmer’s View



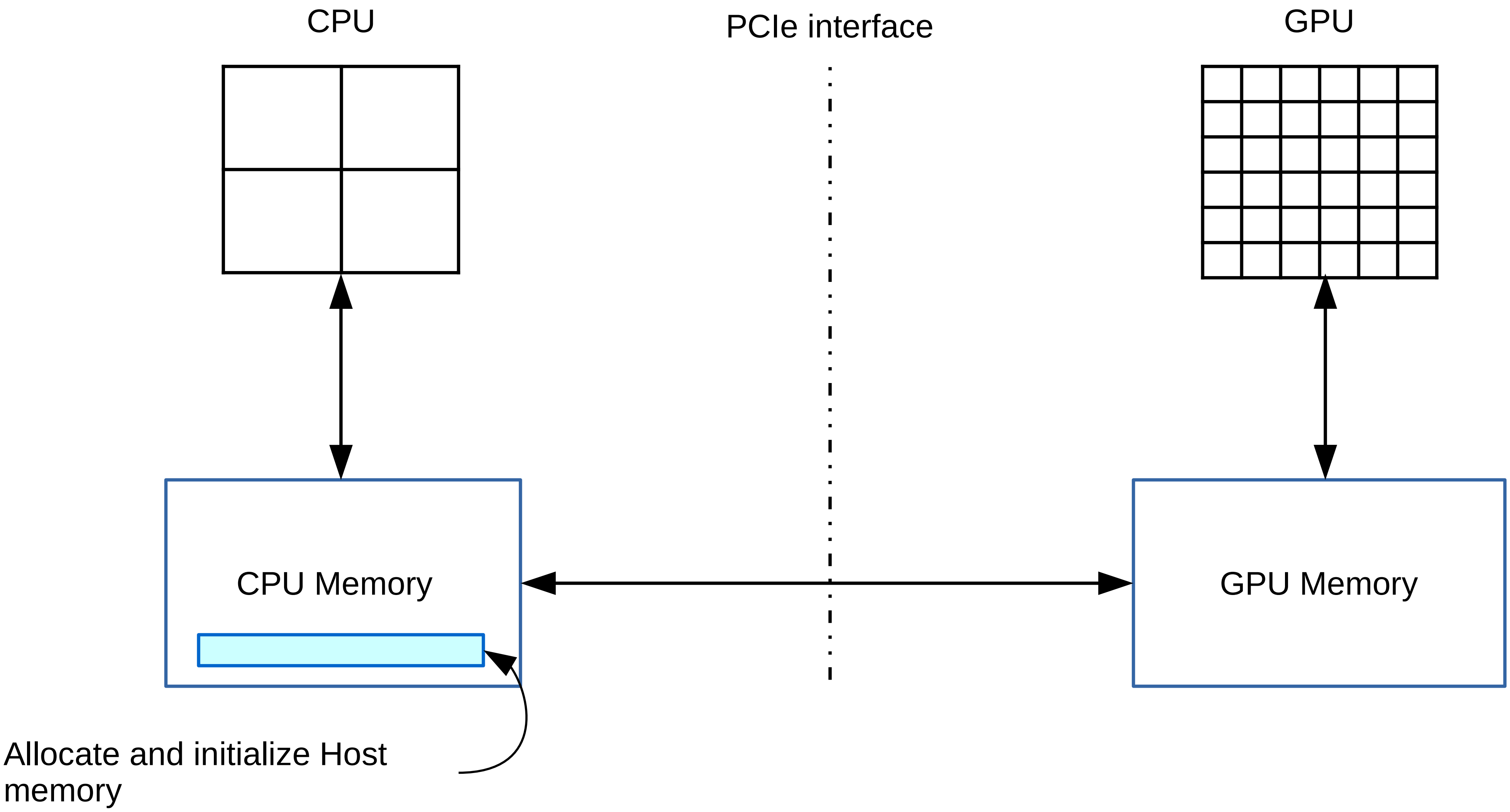
Programming GPUs

- CUDA
- OpenCL
- OpenMP 4.0+
- OpenACC
- . . .

Pseudo Code (old method)

1. Allocate and initialize memory on Host (CPU)
2. Allocate memory on Device (GPU)
3. Transfer data from Host memory to Device memory
4. Launch “kernel” on the Device – large number of threads execute in parallel on Device
5. Transfer results from Device memory to Host memory
6. De-allocate all the memory and terminate the program

Step 1



CPU

PCIe interface

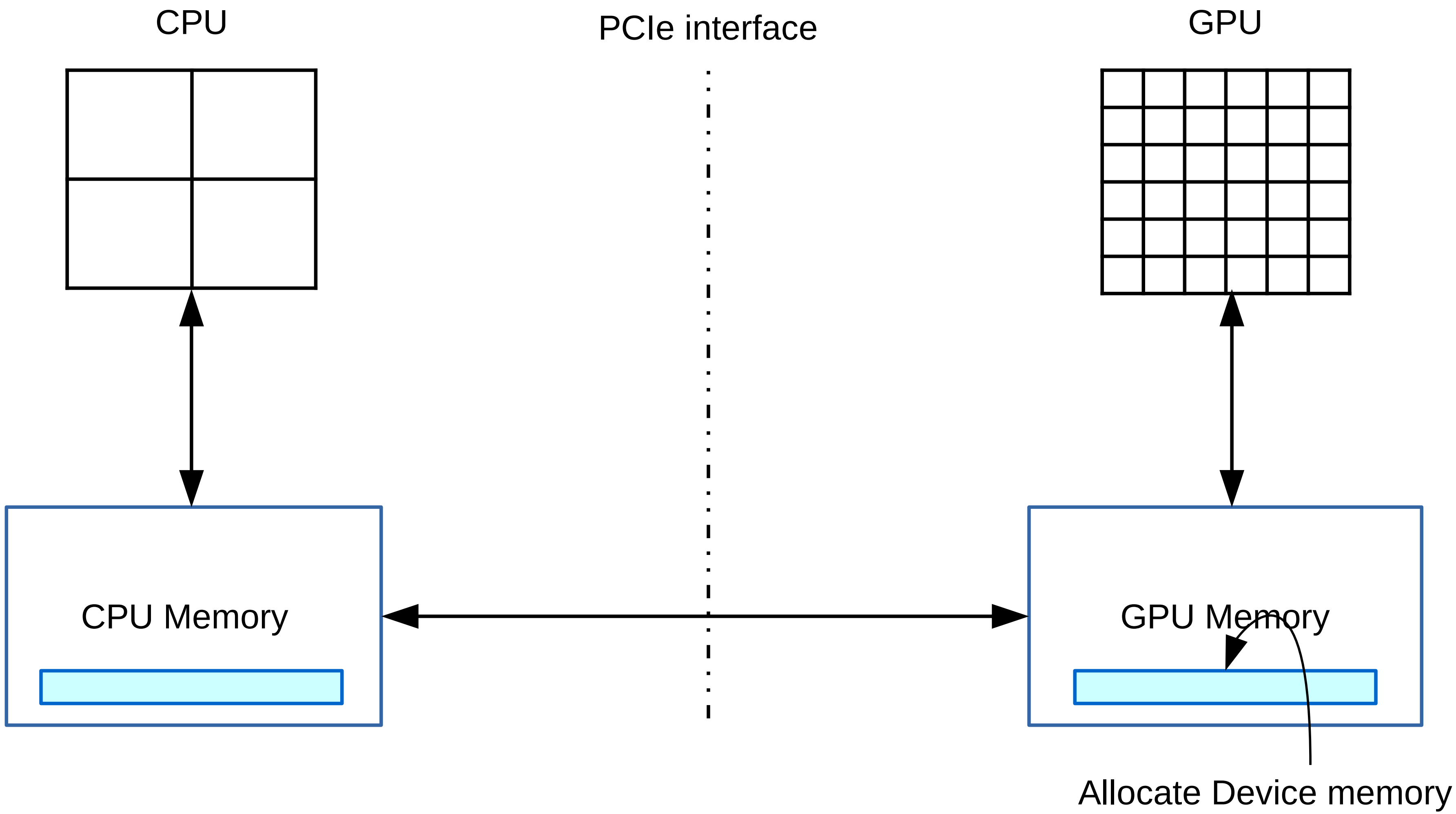
GPU

CPU Memory

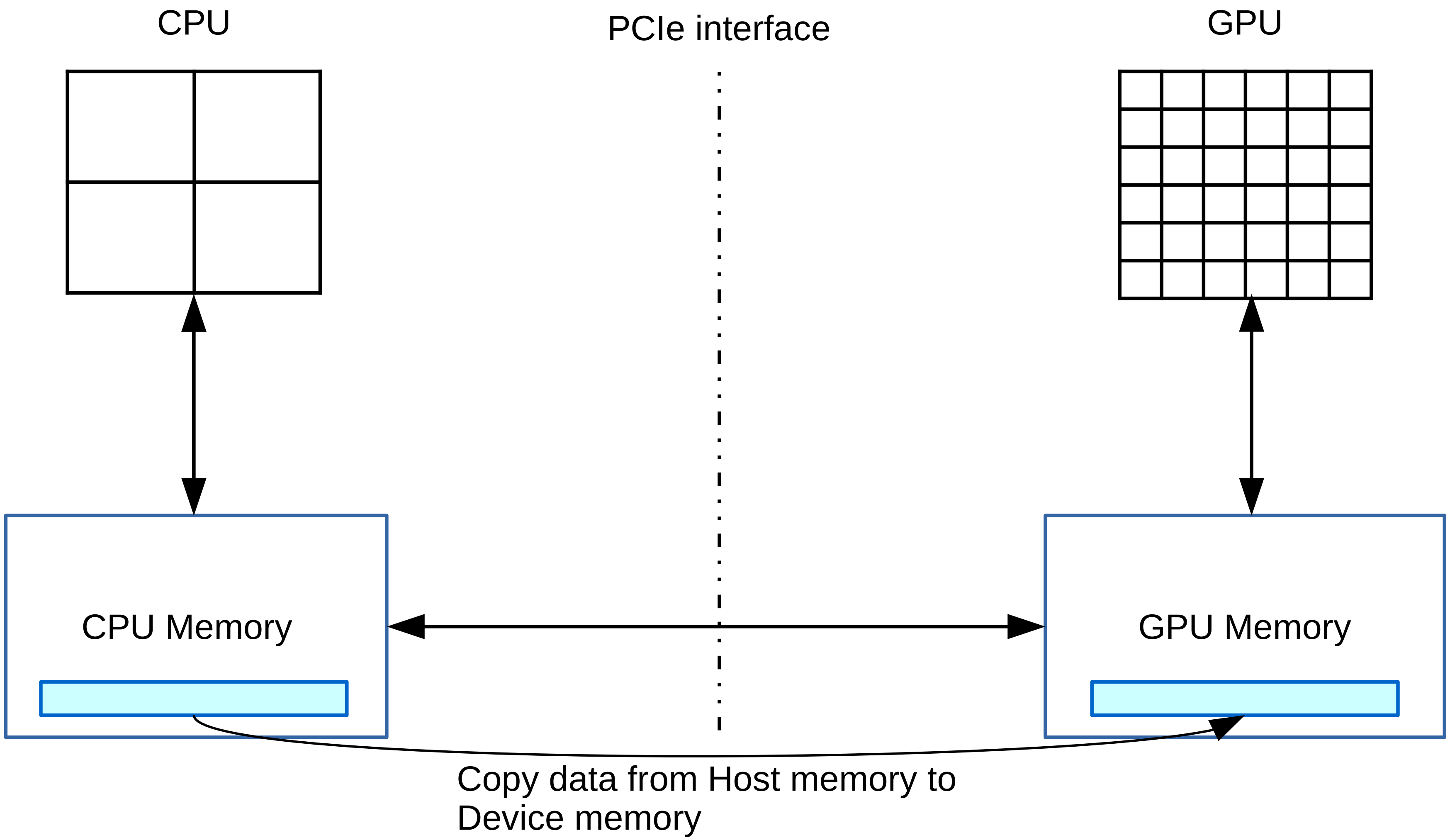
GPU Memory

Allocate and initialize Host memory

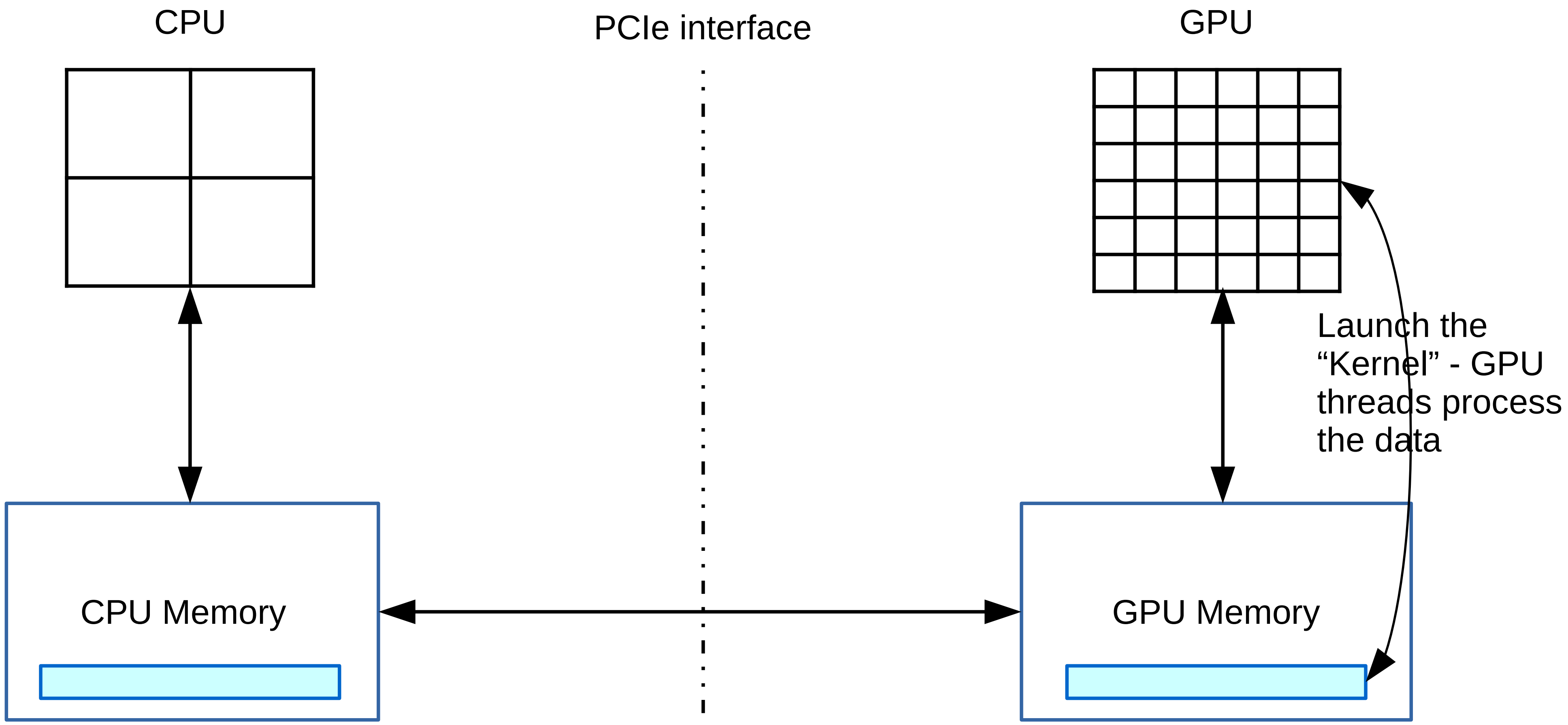
Step 2


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Allocate Device memory

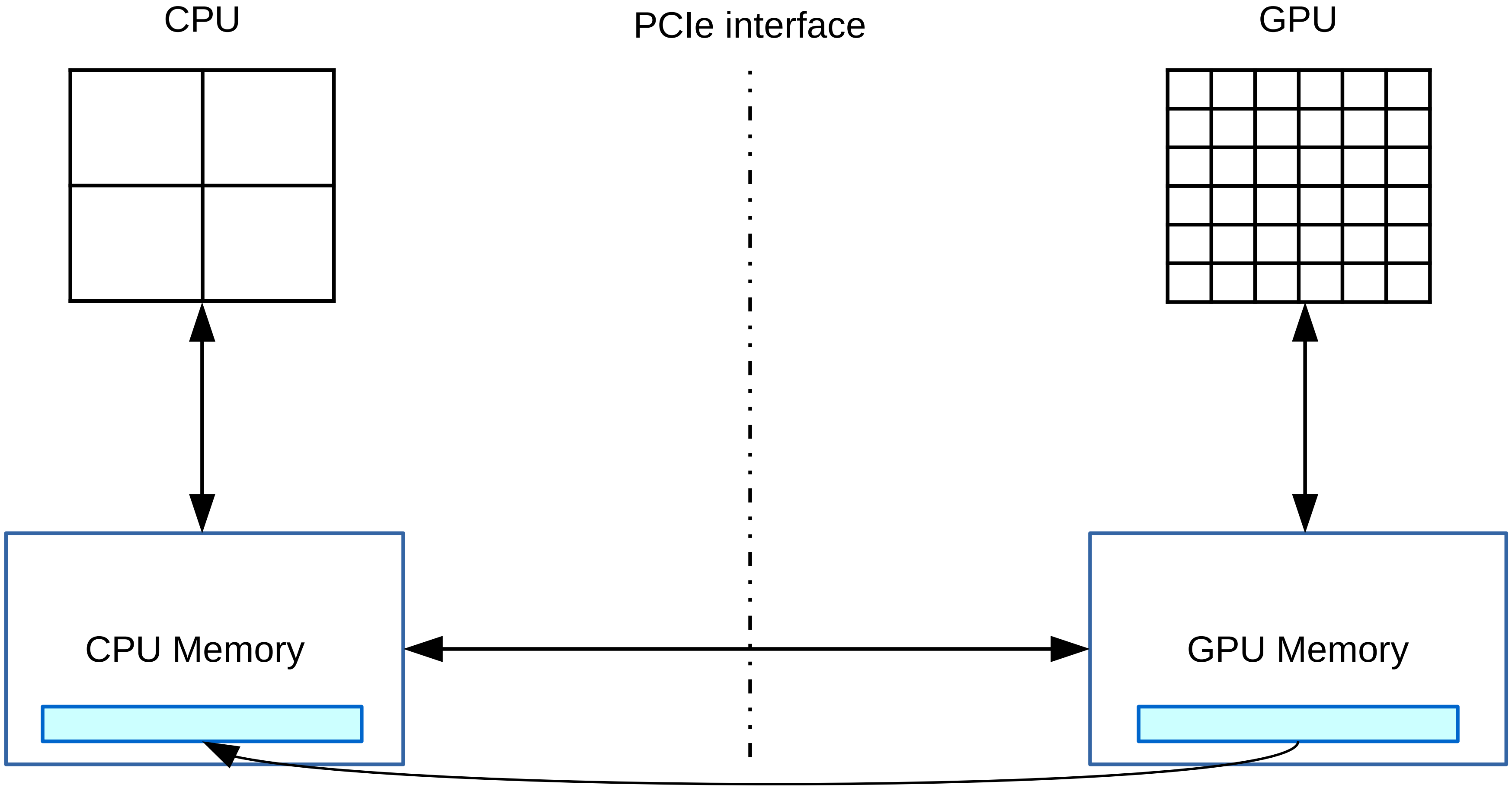
Step 3


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Copy data from Host memory to Device memory

Step 4

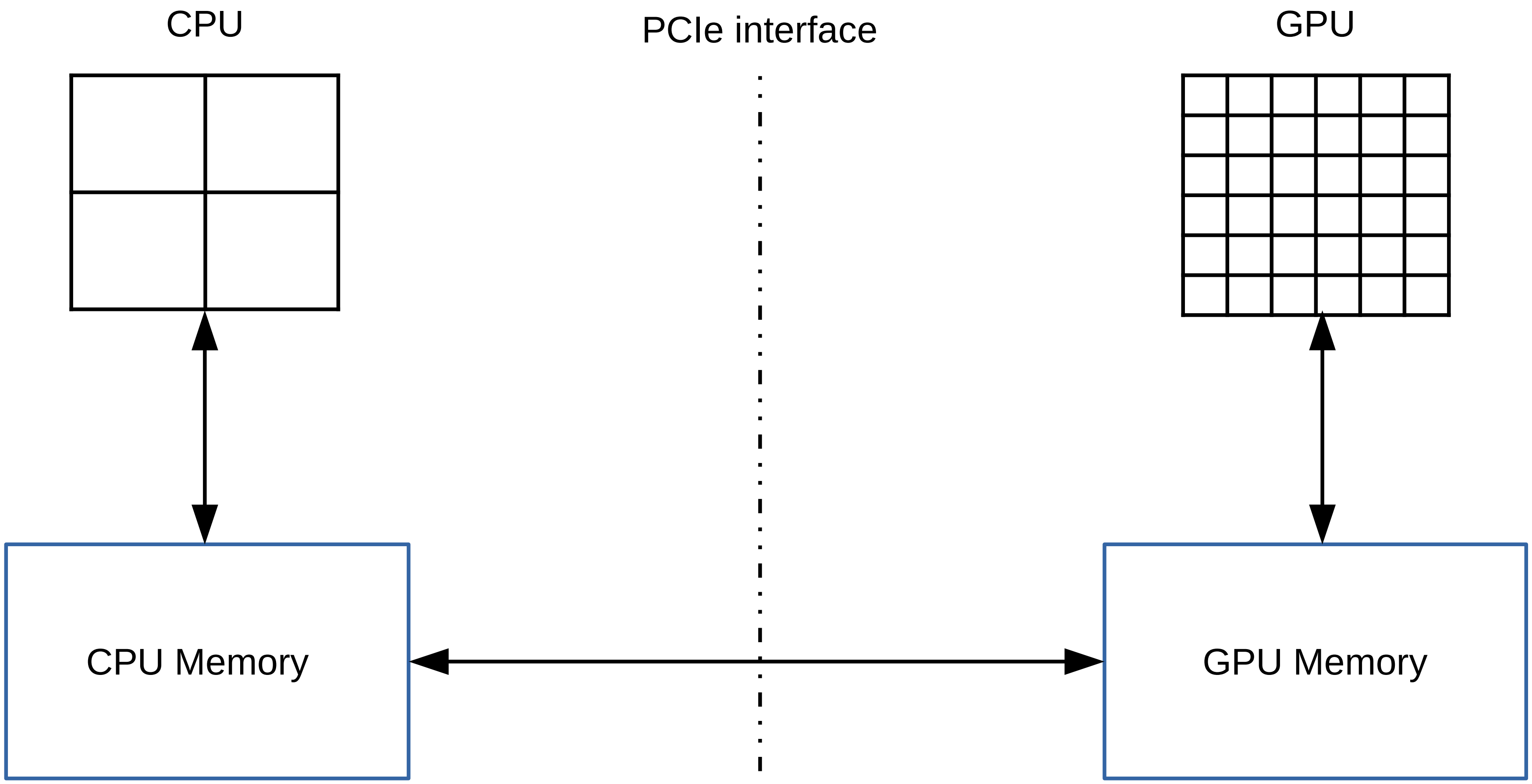

CPU
PCIe interface
GPU
Launch the "Kernel" - GPU threads process the data
CPU Memory
GPU Memory

Step 5



Transfer the results from Device memory to Host memory

Step 6

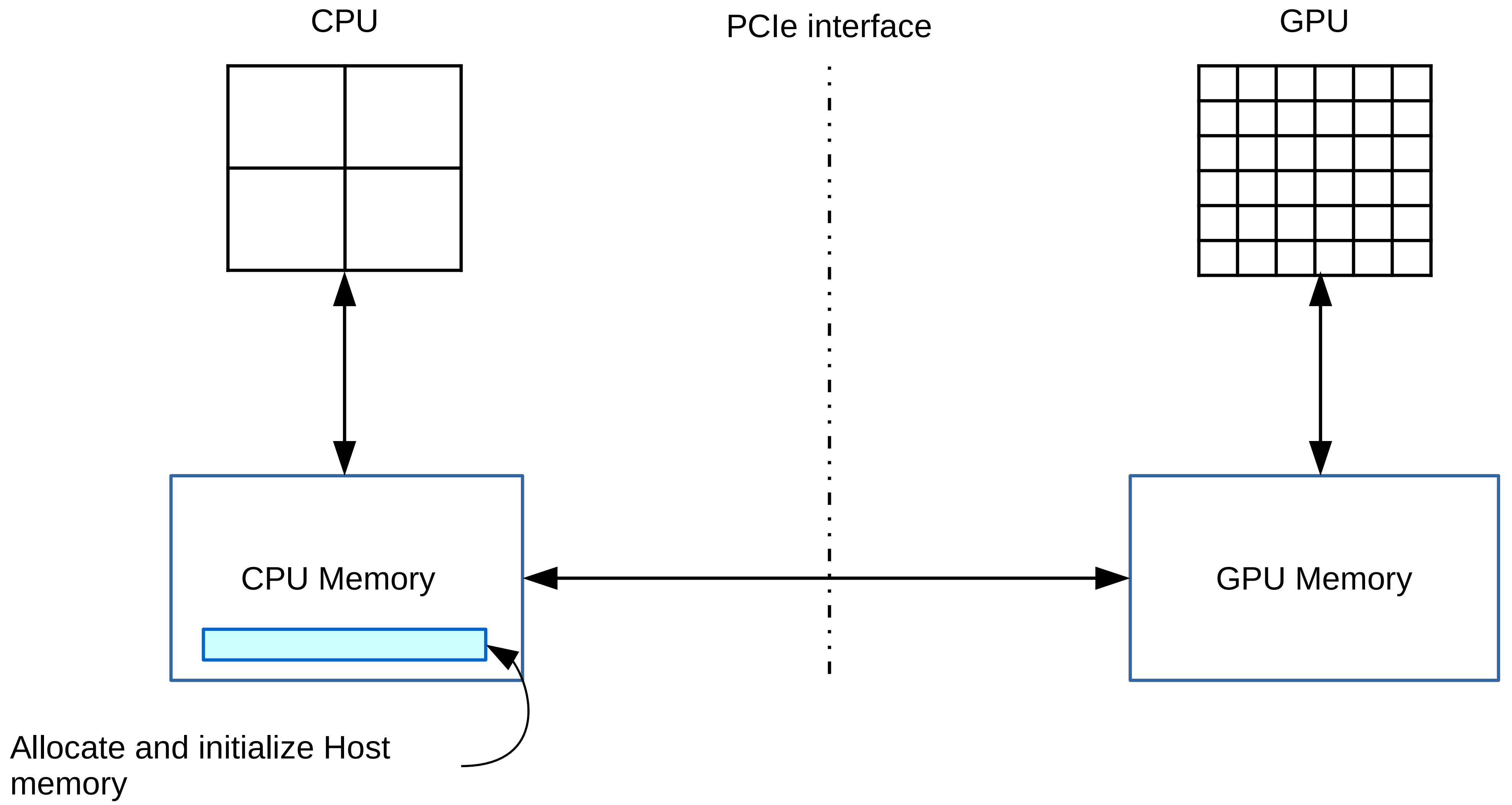


De-allocate the memories and
terminate the program

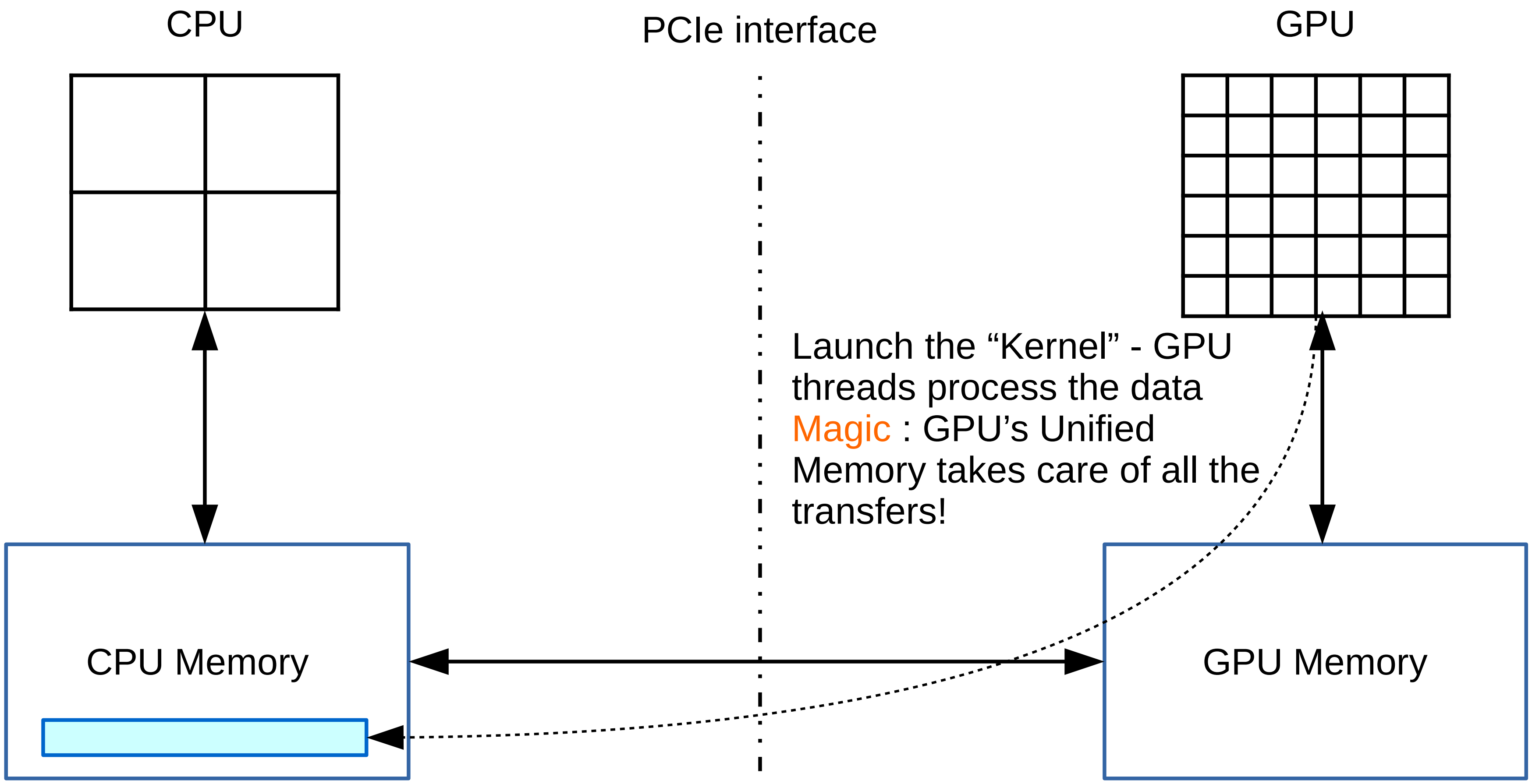
Pseudo Code (new method)

1. Allocate and initialize Host Memory
2. Launch “kernel” on the Device – large number of threads execute in parallel on Device (Magic Step!)
3. De-allocate all the memory and terminate the program

Step 1


CPU
PCIe interface
GPU
CPU Memory
GPU Memory
Allocate and initialize Host memory

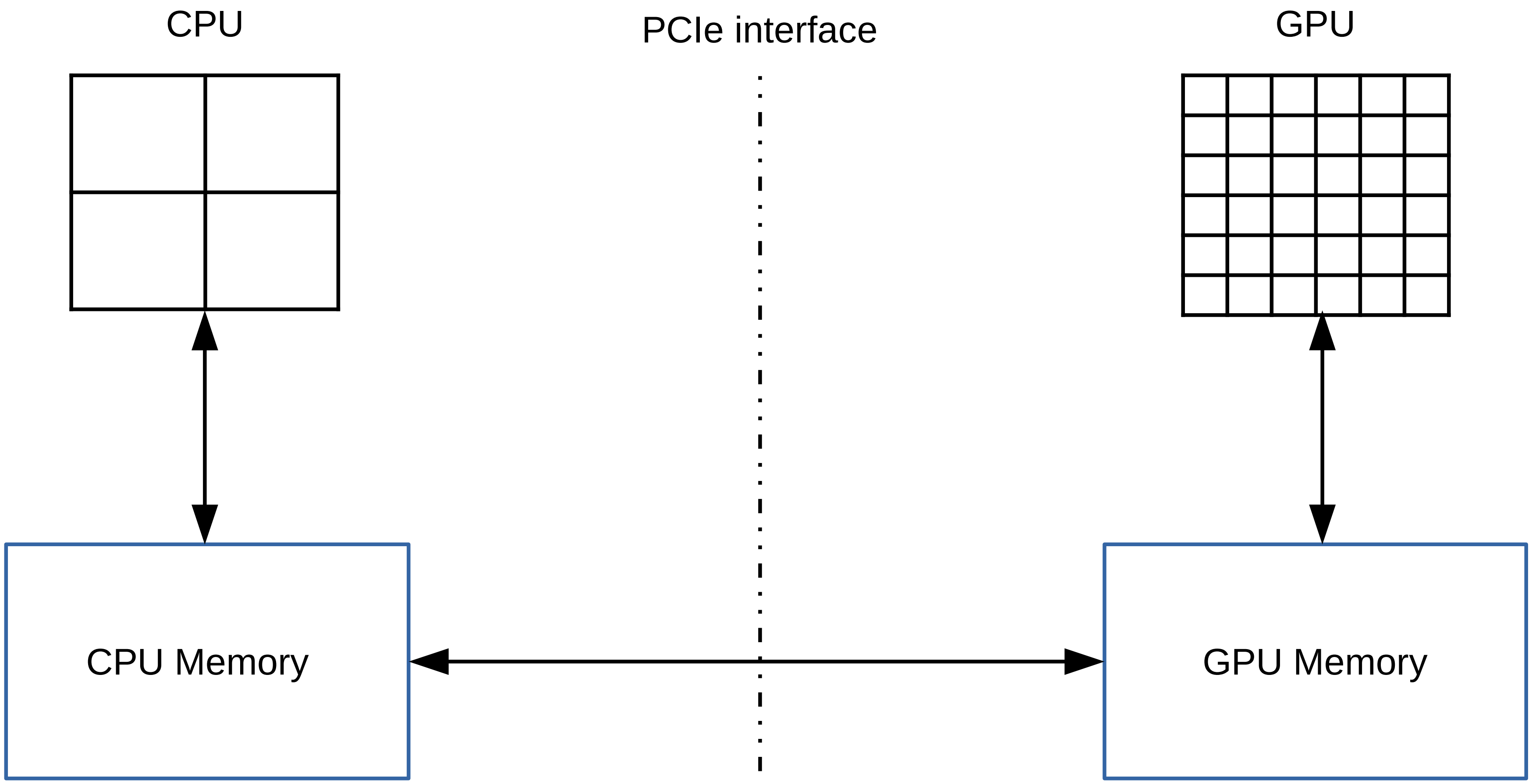
Step 2 (Magic Step!)



Launch the “Kernel” - GPU threads process the data

Magic : GPU’s Unified Memory takes care of all the transfers!

Step 3



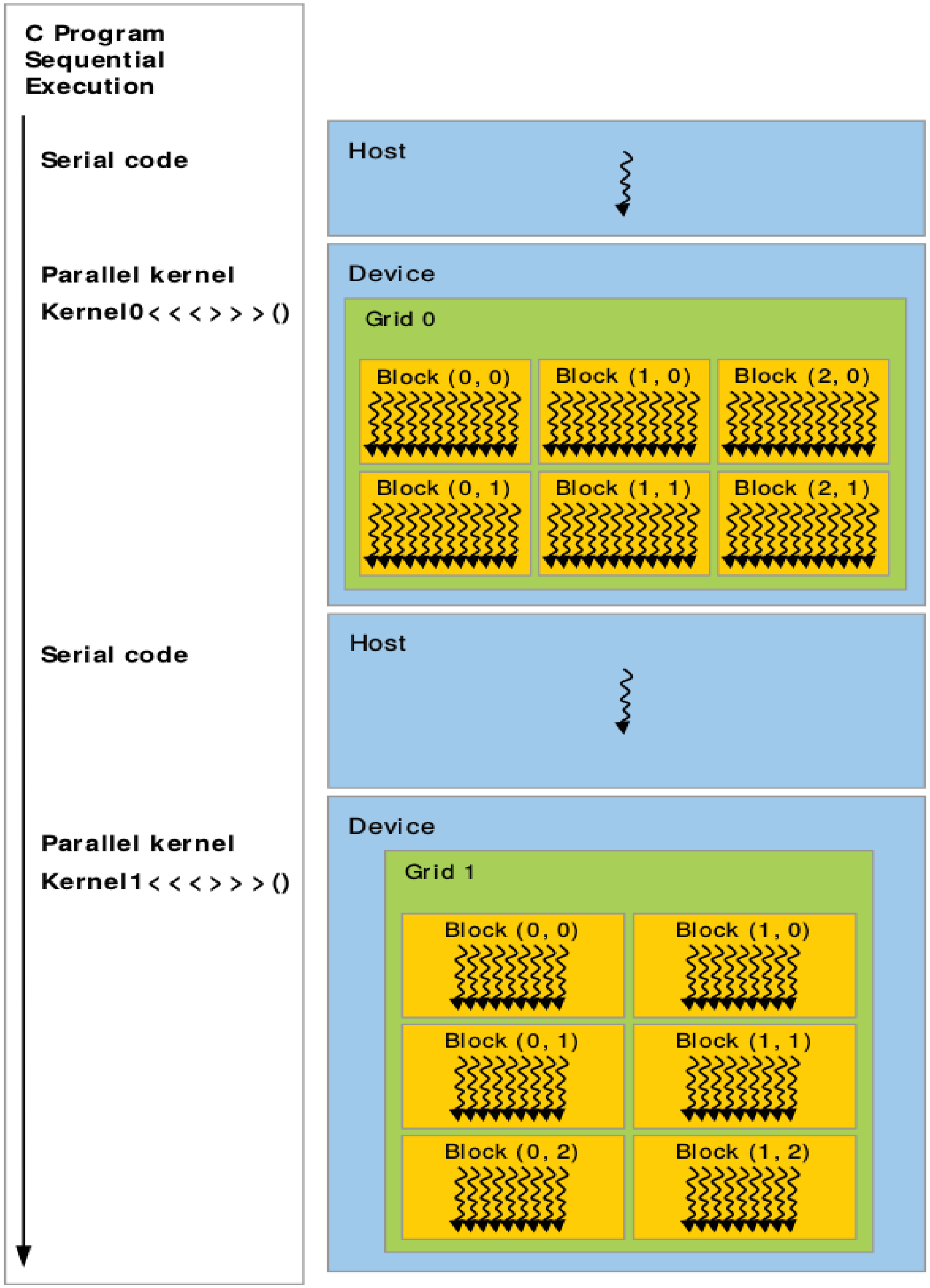
De-allocate the memory and
terminate the program

Introduction to CUDA

CUDA

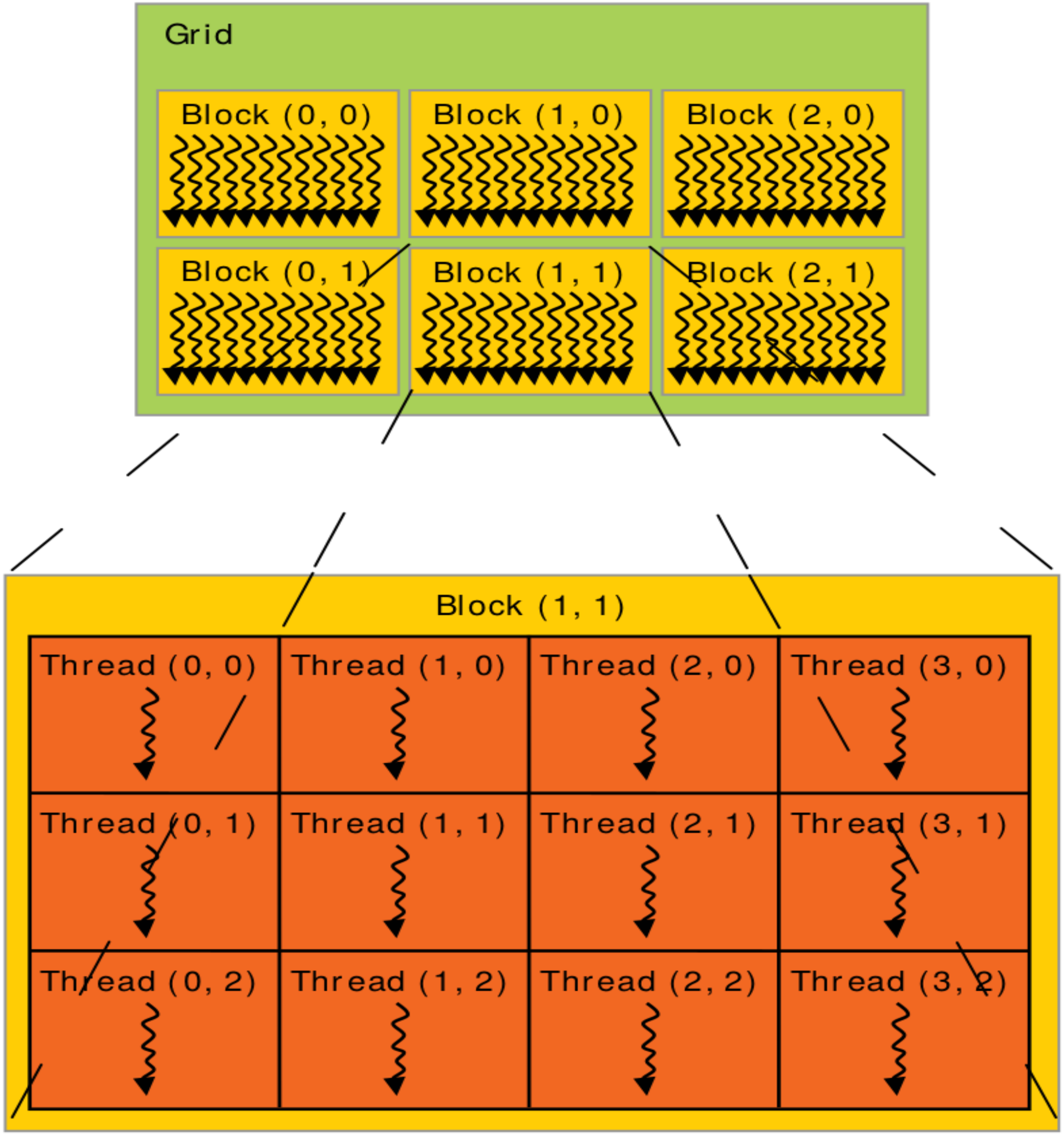
- Parallel computing platform and application programming interface (API)
- Nvidia Corp.- General Purpose Graphics Processing Unit (GPGPU)
- Supports most of the today's Nvidia's gaming cards.
- Platforms
  - Geforce : Desktop
  - Quadro : Workstation
  - Tesla : Datacenter
  - Tegra, Jetson, Drive : Embeded

Structure of a CUDA Code



Source: CUDA C Programming Guide (NVIDIA)

Threads, Blocks and Grid(s)



Source: CUDA C Programming Guide (NVIDIA)

Vector addition (old method)

```
int m[200], n[200], p[200],*md, *nd,*pd;
int size = 200 * sizeof(int);
// Initialize array m and array n
...
cudaMalloc(&md, size);
cudaMemcpy(md, m, size, cudaMemcpyHostToDevice);
cudaMalloc(&nd, size);
cudaMemcpy(nd, n, size, cudaMemcpyHostToDevice);
cudaMalloc(&pd, size);
arradd<<< 1,200 >>>(md,nd,pd);
cudaMemcpy(p, pd, size, cudaMemcpyDeviceToHost);
cudaFree(md);
cudaFree(nd);
cudaFree(pd);
```

“Kernel” function

```
__global__ void arradd(int* md, int* nd, int* pd)
{
int myid = threadIdx.x;

pd[myid] = md[myid] + nd[myid];
}
```

Questions?

Thank you.