Offloading Computation to a GPU with OpenMP

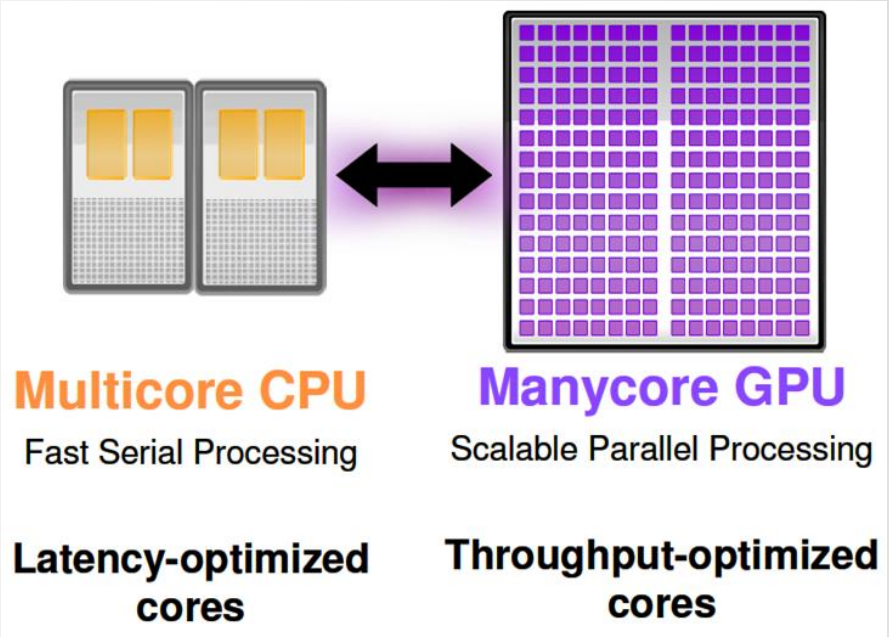
Kamer Kaya, Sabancı University

ncc@ulakbim.gov.tr

CPUs and GPUs

CPU Threads:

- Heavyweight entities managed by the operating system.
- Context switching saves and restores thread states, and this process can be slow and resource-intensive.
- CPU cores are optimized for minimizing latency for one or two threads at a time.



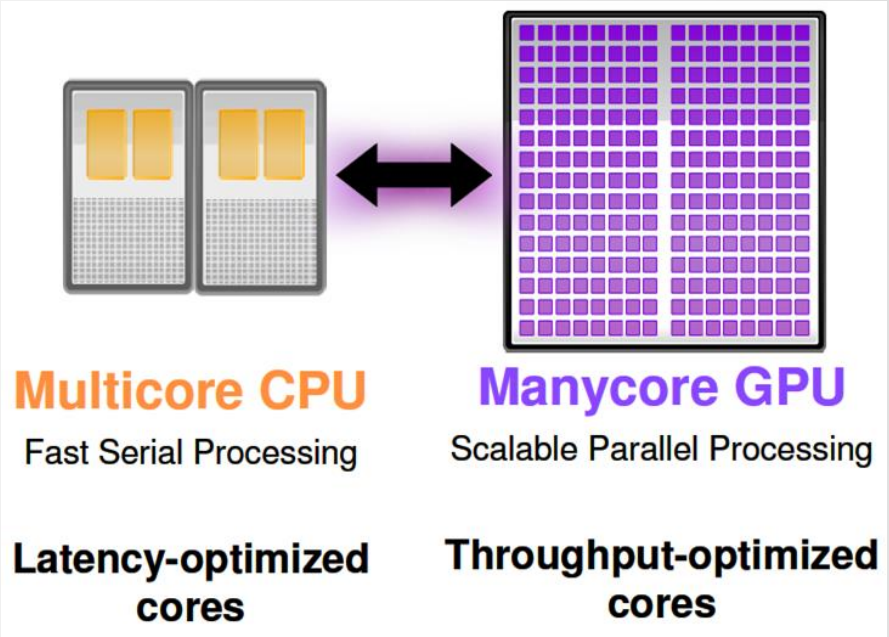
https://tatourian.blog/2013/09/03/nvidia-gpu-architecture-cuda-programming-environment/

CPUs and GPUs

EURO²

GPU Threads:

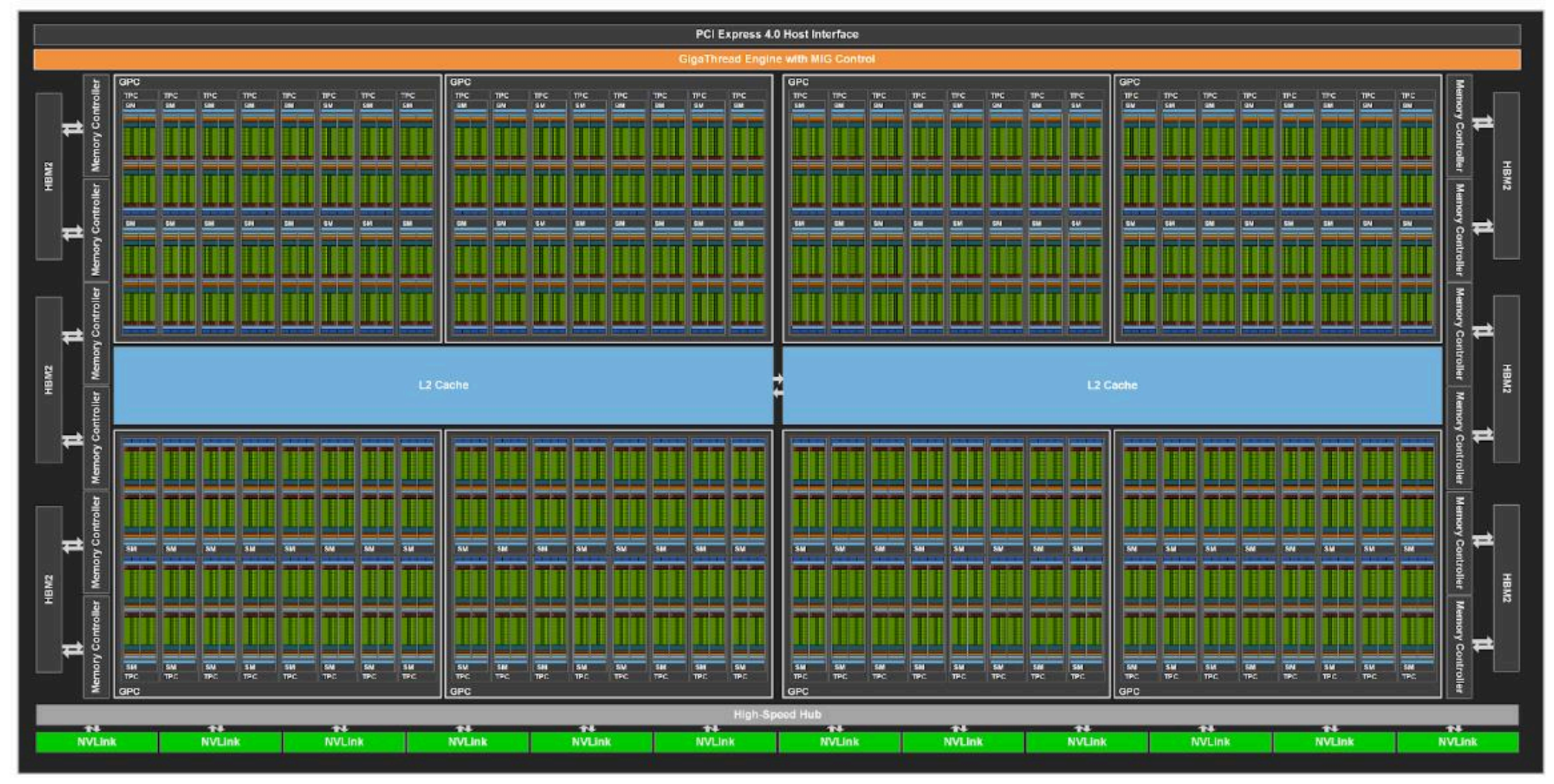
- Lightweight and designed for massive parallelism.
- Hundreds of thousands of threads are grouped into warps of 32 threads each.
- Seamless switching between warps when one is waiting, without saving or restoring thread states.
- Each thread has dedicated registers and resources, which remain allocated until execution completes.



https://tatourian.blog/2013/09/03/nvidia-gpu-architecture-cuda-programming-environment/

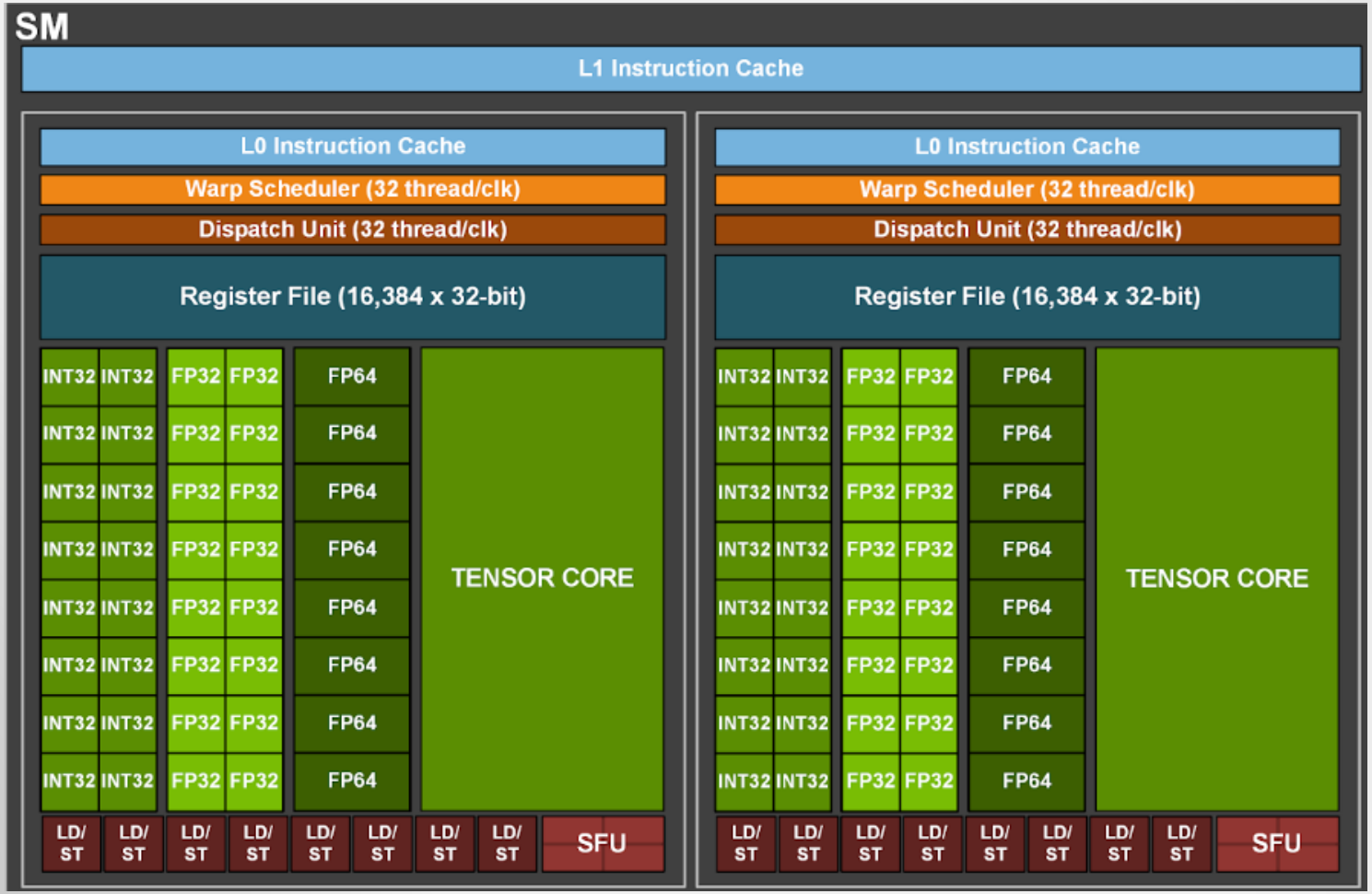
GPU Architecture: Overview

EURO²



https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

GPU Architecture: Overview



https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

Programming GPUs: Benefits

High Parallelism: GPUs can execute thousands of threads simultaneously.
High Throughput: Optimized for tasks involving massive data parallelism.
Applications:

- Graphics rendering
- Machine learning
- Scientific simulations

Example: Computing Pi

$$\pi = 4 \int_0^1 \sqrt{1 - x^2}\, dx$$

Numerical Approximation

To compute this integral numerically:

1. Divide the interval $[0, 1]$ into N small intervals of width Δx.

$$\Delta x = \frac{1}{N}$$

2. Approximate the integral using the midpoint rule:

$$\pi \approx 4 \cdot \Delta x \sum_{i=0}^{N-1} \sqrt{1 - (x_i)^2}$$

where:

$$x_i = (i + 0.5) \cdot \Delta x$$

Example: Computing Pi with CUDA

EURO²

```
__global__ void integrate_pi(double *d_results, int num_intervals, double step) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    double x, local_sum = 0.0;

    for (int i = idx; i < num_intervals; i += gridDim.x * blockDim.x) {
        x = (i + 0.5) * step;
        local_sum += sqrt(1.0 - x * x) * step;
    }

    d_results[idx] = local_sum;
}
```

Each thread executes this code

ID of the current thread (differentiates the execution from those of other threads"

$$\pi \approx 4 \cdot \Delta x \sum_{i=0}^{N-1} \sqrt{1-(x_i)^2}$$

```
// Launch the kernel
integrate_pi<<<NUM_BLOCKS, THREADS_PER_BLOCK>>>(d_results, num_intervals, step);
```

Kernel launch

Example: Computing Pi with CUDA

EURO²

- **Let's work on this CUDA code.**

Thanks

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101101903. The JU receives support from the Digital Europe Programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia