Introduction to Parallel & Distributed Programming

Lec 15— CUDA

Subodh Sharma | March13, 2026

GPU Model of Computation

- Programs runs partly on **Host** (CPU), and partly on **Device** (GPU)
- Host orchestrates the computation
  - Meaning — send computation task and data to the GPU
- Device executes tasks specified as **kernels** in parallel
  - In parallel Kernel execution — > Through non-blocking launches (using **Streams)**
  - **Example Kernel & its launch code**
- Threads smallest unit of execution
- Threads are organised in TBs
  - TB executes on a single SM

```
__global__ void addVectors(float *A, float *B, float *C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

```
addVectors<<<gridSize, blockSize>>>(A,B,C);
```

CUDA Basics

- **Kernel** - The unit of computation for a GPC

```
__global__ void vector_add(const float *A, const float *B, float *C, int N)
{

    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < N)
        C[i] = A[i] + B[i];
}
```

Explain why this check is required

Device Qualifier: On device but called from the host

CUDA Basics

Device Qualifiers

- **__device__ :** Called from the device and executes in the device
  - Used for helper functions
- **__host__:** Called from and executes on the host
  - Standard C++ functions
- **__global__:** On device called from the host
  - Defines a kernel

CUDA Basics

Variable Qualifiers

- __device__ type var_name;
  –resides in GDRAM, scope is lifetime of the app

- __constant__ type var_name;
  –resides in constant memory space on the device
  –scope is the lifetime of the application

- __shared__ type var_name;
  –resides in the shared memory space of the thread block
  –scope is the lifetime of the block

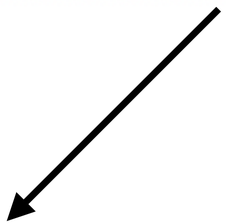
CUDA Basics

- **Kernel Invocation (from a HOST function)**

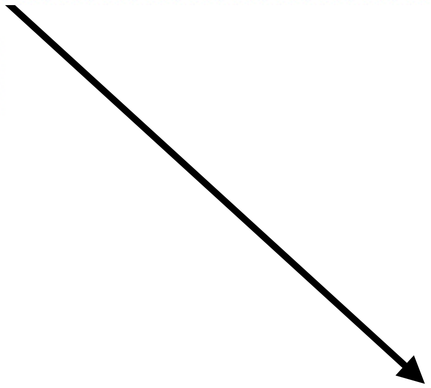
```
int threads_per_block = 256;
int blocks = (N + threads_per_block - 1) / threads_per_block;

vector_add<<<blocks, threads_per_block>>>(d_A, d_B, d_C, N);

CUDA_CHECK(cudaDeviceSynchronize());
```



Wait for GPU to finish computation



Grid Dimension, Block Dimension

CUDA Basics

- **Data Transfer** **(from HOST to DEVICE)**

```
float *d_A, *d_B, *d_C;
CUDA_CHECK(cudaMalloc(&d_A, bytes));
CUDA_CHECK(cudaMalloc(&d_B, bytes));
CUDA_CHECK(cudaMalloc(&d_C, bytes));

// ── Step 3: Copy input data from CPU → GPU ──────
CUDA_CHECK(cudaMemcpy(d_A, h_A, bytes, cudaMemcpyHostToDevice));
CUDA_CHECK(cudaMemcpy(d_B, h_B, bytes, cudaMemcpyHostToDevice));
printf("Copied A and B: CPU → GPU\n");

... (kernel invocation) ..

CUDA_CHECK(cudaMemcpy(h_C, d_C, bytes, cudaMemcpyDeviceToHost));
```

Heap allocation on the DEVICE

Direction of Memcpy — H2D

Next CUDA Topics

- Shared memory usage and CUDA Synchronisation primitives
- CUDA programming pitfalls
- CUDA optimisations