CUDA Programming and GPU Architecture

Chris Kauffman

Last Updated:
Tue Nov 16 10:18:01 PM CST 2021

Logistics

GPUs will Feel Different

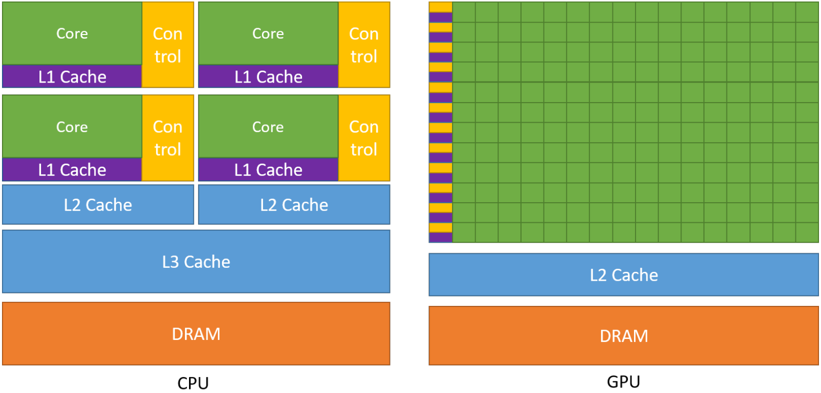
Distributed / Threaded Programming

- Most effective strategies looked for ways to assign lots of work to limited number of procs/threads
- Poo-pooed the idea of "Assume length N array and N processors"

GPU Programming

- Threads are essentially cost-free, close to theoretical models so...
- Assume length N array and N processors
- Will require some mental adjustment

GPU vs CPU



Source: NVidia Docs “CUDA C++ Programming Guide”

GPUs are a Co-Processor or Accelerator

- CPU is still in charge, has access to main memory
- GPU is a partner chip, has a distinct set of memory
- Sections of code will feel like Distributed architecture
  - CPU / GPU memory transfers
  - Barriers / synchronization as CPU waits for GPU to finish
- GPU itself is like a multicore system on steroids

Why do GPUs Look like this?

CUDA : NVidia's General Purpose GPU Technology

CUDA Terminology

Thread A set of operations; can be as small as a single addition

Kernel A function which expresses what a thread should do

Block A group of executing threads which can share some local memory

Execution Context Run a Kernel function with a specified number of Blocks, Threads per Block, and amount of shared memory

Host The CPU - sets Execution Context, launches Kernels on GPU

Device The GPU which runs Kernels

Hello CUDA

Examine `hello.cu`: `.cu` extension favored for CUDA programs; it is C++ w/ a few extras

Kernel

```
__global__ void helloFromGPU() { // __global__ => called from CPU/GPU
                                 // runs on GPU

  printf("Block %02d Thread %02d: Hello World\n",
         blockIdx.x,            // ever-present structs which gives
         threadIdx.x);          // each GPU thread indexing info

}
```

Execution Context

```
int main (int argc, char *argv[]){
  printf("CPU: Running 1 block w/ 16 threads\n");
  helloFromGPU<<<1,16>>>();     // executes in 1 block, 16 threads per block
  cudaDeviceSynchronize();      // ensures GPU completes operations
```