CS 220: Introduction to Parallel Computing

CUDA

Lecture 29

Today's Schedule

- GPUs vs CPUs
- More Terminology
- Getting Started Lab
- Last 15 minutes:

  Please fill out teaching effectiveness surveys

  (you should've been emailed a link)

Today's Schedule

- **GPUs vs CPUs**
- More Terminology
- Getting Started Lab
- Last 15 minutes:

  Please fill out teaching effectiveness surveys

  (you should've been emailed a link)

GPUs vs CPUs

- We discussed the differences between GPUs and CPUs last class
  - CPU = single paintball gun
  - GPU = lots of paintball guns?!
- To be more fair: the CPU offers us a lot more flexibility, but fewer cores
- Naturally, these differences lead to changes in how we think about implementing our programs

GPUs vs CPUs

- So we need to think about our problems in a different way…
- MPI is similar: many times we might think “we need to do this in a loop”
  - …but we actually don’t always need loops, since all the processes will execute the same thing

Today's Schedule

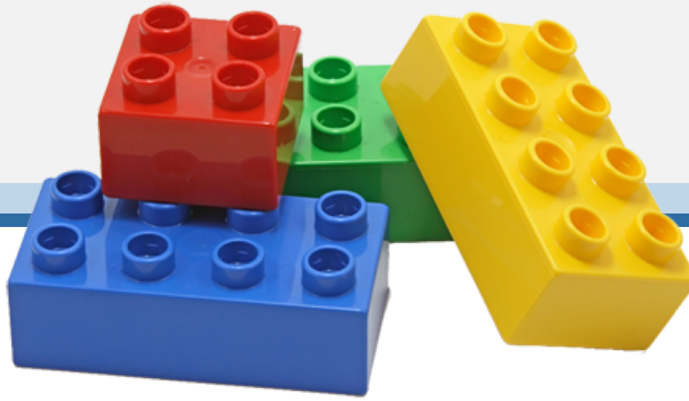
- GPUs vs CPUs
- **More Terminology**
- Getting Started Lab
- Last 15 minutes:

  Please fill out teaching effectiveness surveys
  (you should've been emailed a link)

The Kernel

- When your CUDA program starts, it executes the main() function on the **host**
  - Runs on the CPU and uses main memory
- The host is responsible for launching the **kernel**
  - Runs on the GPU and uses GPU memory
- Kernels are launched with this syntax:
  kernel_name<<<grid_size, block_size>>>(params);
  “Triple chevron” syntax <<< >>>

Grids and Blocks



- A **grid** is a collection of **blocks**
- A **block** is a collection of **threads**
- Total number of threads?

  grid_size * block_size
- Note: the blocks can also be 2D and 3D:

  <<<grid_sz, X, Y, Z>>>

Why it Matters

- Grids, blocks, and threads closely follow the hardware and logical design of GPUs
  - Each block gets assigned to a streaming multiprocessor (SM)
  - Memory availability/speed is impacted by thread organization
- By configuring these parameters correctly, we can get better performance
- For instance…

Performance Considerations

- Each block gets assigned to an SM
- The SMs split their blocks into **warps**
  - CUDA unit of SIMD execution
  - A warp = 32 threads
- If the number of threads in the block isn't evenly divisible by 32, then we'll have inactive threads:
  - 20 threads? 12 are inactive

Hello World

- Let’s take a look at doing our usual “hello world” (CUDA style)

Math

- So, CUDA is not great for printing strings!
- More recent hardware and versions of the SDK do support printing directly from the kernel
- cuPrintf is another option
- Let's try something that GPUs are better at: math

Argument Passing

- Arguments to the CUDA kernel are passed by value (copied)
- To return anything back to the host program, we have to use:

    cudaMalloc

    cudaMemcpy

  And our old friend, pointers!

Today's Schedule

- GPUs vs CPUs
- More Terminology
- **Getting Started Lab**
- Last 15 minutes:

  Please fill out teaching effectiveness surveys

  (you should've been emailed a link)

nvcc

- MPI and pthreads are both **libraries** built on C
- CUDA requires its own compiler
  - We're not writing standard C/C++ code
- A CUDA program ends with the extension **.cu**
- Compiling:

```
nvcc my_program.cu -o my_program
```

- To use nvcc, we need to configure our systems first

Setting up the Environment

- Every process inherits its **environment** from its parent process
- The environment contains several **environment variables** that contain configuration information
- Want to see your environment variables? Run ‘env’ at your shell prompt
- To use nvcc, we need to set up one variable in particular: PATH

PATH

- To check your path, run:

  ```
  echo $PATH
  ```

- If /usr/local/cuda/bin isn't in the path, you can add it with:

  ```
  export PATH=/usr/local/cuda/bin:$PATH
  ```

- (You can also add this line to your ~/.bash_profile)
- If your path is configured properly, you can run 'nvcc' and will be prompted to provide input files

LD_LIBRARY_PATH

- If you're having problems with CUDA functions not being found, you may also need to modify your LD_LIBRARY_PATH:
- export LD_LIBRARY_PATH=/usr/local/cuda/lib64: $LD_LIBRARY_PATH

Lab: CUDA

- This lab has three parts:
  - In the first part, you'll compile Nvidia's device query program
    - This lets us find out warp sizes, thread info, etc.
  - Second part: vector addition using CUDA
  - Third part: matrix multiplication using CUDA

Today’s Schedule

- GPUs vs CPUs
- More Terminology
- Getting Started Lab
- **Last 15 minutes:**

  Please fill out teaching effectiveness surveys

  (you should’ve been emailed a link)