ECE/CSC 506: Architecture of Parallel Computers
Program 1: GPU Programming
Due: Friday, May 23, 2025

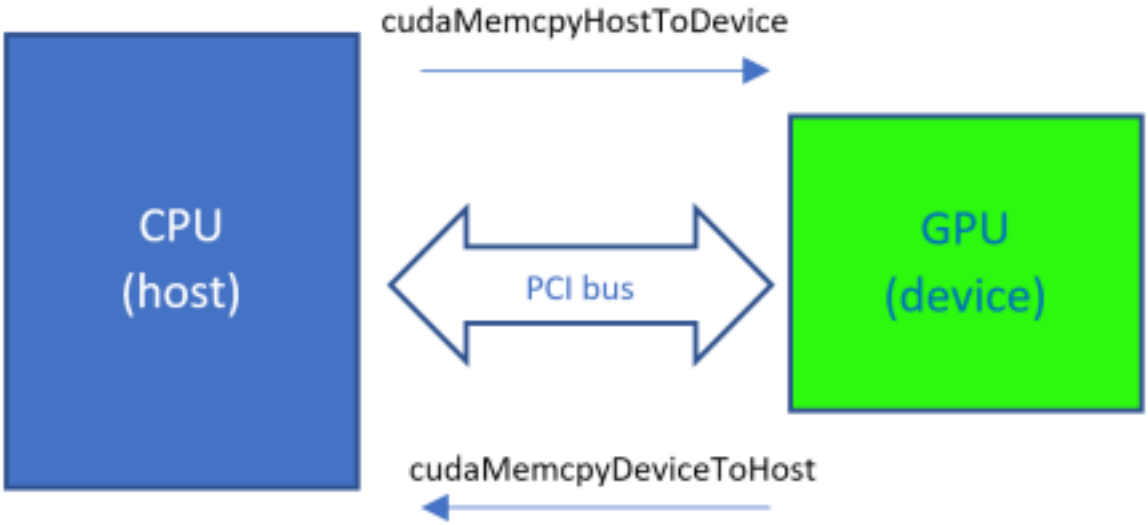
Introduction

GPUs outperform CPUs on applications like image processing, array processing, and plotting graphs. This advantage derives from GPUs' ability to process data in parallel rather than serially.

This assignment asks you to code vector calculations for a graph plot using CUDA code. This assignment has three objectives:

- To introduce you to CUDA.
- To introduce you to simulation environments
- To demonstrate how small modifications in the code can reduce the number of instructions processed by the processor, the number of cache accesses, and overall energy usage.

Background



GPUs generally require a host device to activate them. The CPU (host) will run most of the serial code, but for the parallelizable code, it invokes the GPU kernel.

Note these attributes of CUDA code:

- The kernel invocation is identified by `<<< blocksPerGrid, threadsPerBlock >>>`
- The GPU kernel function header is

```
__global__ ⟨ret type⟩⟨function name⟩(args....)
```

- `cudaMemcpy` is used to copy the data in between device and host

Watch the following videos from the Heterogeneous Parallel Processing course to learn more about programming the GPU. You may take the embedded quizzes, but you are not required to. They will not be counted toward your grade.

a. CUDA memory allocation and data movement API functions [19:37] **Watch**
b. CUDA kernel-based SPMD parallel programming [19:09] **Watch**
c. Kernel-based parallel programming, multidimensional kernel configuration [16:20] **Watch**
d. Kernel-based parallel programming, basic matrix-matrix multiplication [17:17] **Watch**

Environment setup

To see the impact of the four parameters mentioned above on your code, we use `gpgpusim` to simulate the working of the GPU. This simulator simulates the working of the **NVIDIA GeForce GTX 480**. To set up the environment on your PC, follow these steps:

1. Get the docker container for gpgpu-sim by following the instructions in the Docker setup 506 file.

NOTE: Do not update any installed packages in the Linux environment. The simulator requires a certain version of packages (e.g., `gcc`) to function smoothly. Kindly leave the packages untouched.

Then click on the ece506_gpgpu-sim button at the left and then click the "Start" arrow and perform a "normal start".

4. Launch a terminal and run the following commands once you are in the docker container for gpgpusim:

```
cd ~/gpgpu-sim_distribution
source setup_environment  (This will source the environment file)
make clean
make                (compiles and builds the gpgpu-sim)
git clone https://github.com/peiyi1/vectorAdd.git
cd ~/vectorAdd  (change to the cloned directory)
make clean
make                (this will compile the vectorAdd.cu file)
cp ~/gpgpu-sim_distribution/configs/tested-cfgs/SM2_GTX480/* .  (copy config files)
./vectorAdd
```

Wait for the completion of the run of the simulator. This may take some time. If the output says "Test PASSED", the environment works for CUDA programs, and you're good to go! If not, repeat the steps.

5. Read the vectorAdd.cu code given in the `vectorAdd` directory. Understand the memory allocation, copying of data and memory freeing involved in the code. Also, understand how the CUDA kernel function works.

Experiment

Download the `program1.zip` file from the assignment resources, which has `plot1` and `plot2` as subfolders. You need to unzip it and replace the submission folder in the Ubuntu VM image with it.

Examine the `VectorAdd` folder in the home directory. It will help you understand the basics of CUDA programming.

Using the `Makefile` inside the `vectorAdd` folder as a reference, create your own `Makefile` for the folders `plot1` and `plot2`.

Now, program the GPU kernel to implement the following equations:

Plot 1:

- Vector element-wise operation.
- $C(x) = (2*A(x)^4 + 5*A(x)^2\,B(x)^2 + 3*B(x)^4)/(D(x)^2)$
- Requirements:
  - implement an optimized kernel.
  - add appropriate code to accommodate vector D.

Plot 2:

- Matrix operation.
- *A, B, C, D* are all matrices
- *AB* and *AD* are matrix multiplication.
- $C = 3AB + 4AD$
- Requirements:
  - implement an optimized kernel.
  - add appropriate code to accommodate matrix D.

To test your optimized kernel, modify the value of OPT, rebuild and run.

When running the simulator, redirect its standard output to a txt file and include it in the final submission. You need to collect data for both the base kernel and your optimized kernel.

From the simulator output, note the following for the analysis:

1. **Total instructions processed** (gpu_sim_insn)
2. **Total number of cache accesses** (L1I_total_cache_accesses+ L1D_total_cache_accesses) and 3. **Average energy usage** from the power report generated (note that power report log file will be created when you run the code. you'll need to check the power report every time you run the code to get the reading). *Energy = power × number of cycles/clk_frequency*

NOTE: You are free to consider other details of the output as well in your analysis.

Plot1	**Base code**	**Energy Optimized code**	**% improvement**
Total instructions processed			
Total number of cache accesses			
Average energy usage			

Two tables need to be created, one for `plot1` and one for `plot2`.

Analysis

The analysis should be done per plot. You need to compare the base code results with the energy efficient code results and explain why you observe the changes.

1. Why is the total number of instructions less than in the base code?
2. Why are there fewer cache accesses?
3. What is the parameter that is compromised for saving energy? Why?
4. Which code would you prefer to use, and under which circumstances?
5. How did you find your first experience with a simulator?
6. Where do you think simulators will be helpful?

Submission format

Your submission should contain two folders and a report file in a zip archive. Inside each plot folder, rename the redirected output file and the generated `gpgpu_power_report_xxxx` file to reflect which kernel was used, exactly as shown below. The whole folder structure should look like the following:

- `Plot1`
  - `plot1_base.txt`
  - `plot1_base_power.log`
  - `plot1_opt.txt`
  - `plot1_opt_power.log`
  - `plot1.cu`
  - `config_fermi_islip.icnt`
  - `gpgpusim.config`
  - `gpuwattch_gtx480.xml`
  - `Makefile`
- `Plot2`
  - `plot2_base.txt`
  - `plot2_base_power.log`
  - `plot2_opt.txt`
  - `plot2_opt_power.log`
  - `config_fermi_islip.icnt`
  - `gpgpusim.config`
  - `gpuwattch_gtx480.xml`
  - `Makefile`
- `report.pdf`

Each folder should contain a `.cu` source file and a `Makefile`. Your program should be able to compile when "`make`" is issued in that folder.