Homework 5 - GPU programming

Due: March 26 @ 4:55pm on Gradescope
Deliverables: Code and writeup

This homework is a hands-on introduction to parallel programming on GPUs via CUDA.

Your homework should be completed **individually** (not in groups).

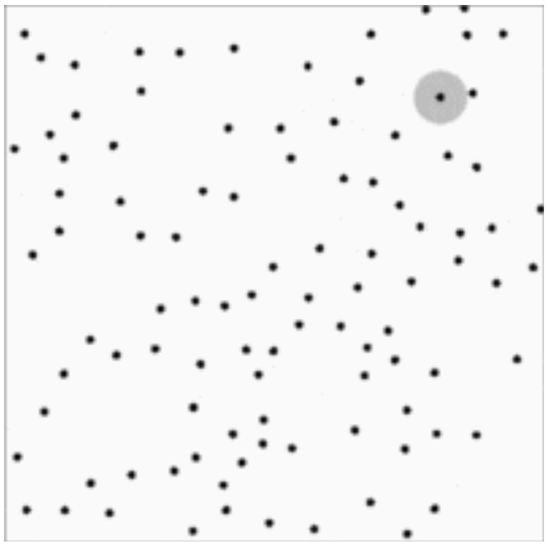
Code and some parts of the handout are from UC Berkeley CS267.

All instructions in this handout assume that we are running on **PACE ICE**. Instructions for logging in are in HW1.

Please focus on parallelizing your code using only the GPU, thus only 1 CPU core and 1 CPU thread (no MPI or OpenMP).

Overview

In this assignment, we will be parallelizing a toy particle simulation (similar simulations are used in mechanics, biology, and astronomy). In our simulation, particles interact by repelling one another. A snapshot of our simulation is shown here:



The particles repel one another, but only when closer than a cutoff distance highlighted around one particle in grey.

Getting and running the code

Download from git

The starter code is available on Github and should work out of the box. To get started, we recommend you log in to PACE ICE and download the code:

```
$ git clone https://github.com/cse6230-spring24/hw5.git
```

There are five files in the base repository. Their purposes are as follows:

CMakeLists.txt
The build system that manages compiling your code.
main.cu
A driver program that runs your code.
common.h
A header file with shared declarations
job-gpu
A sample job script to run the gpu executable
gpu.cu - - - You may modify this file.
A skeleton file where you will implement your gpu simulation algorithm.

Please do not modify any of the files besides gpu.cu.

Build the code

First, we need to set up the modules:

```
$ module purge
$ module load cmake
$ module load cuda/11.7.0-7sdye3
$ module load gcc
```

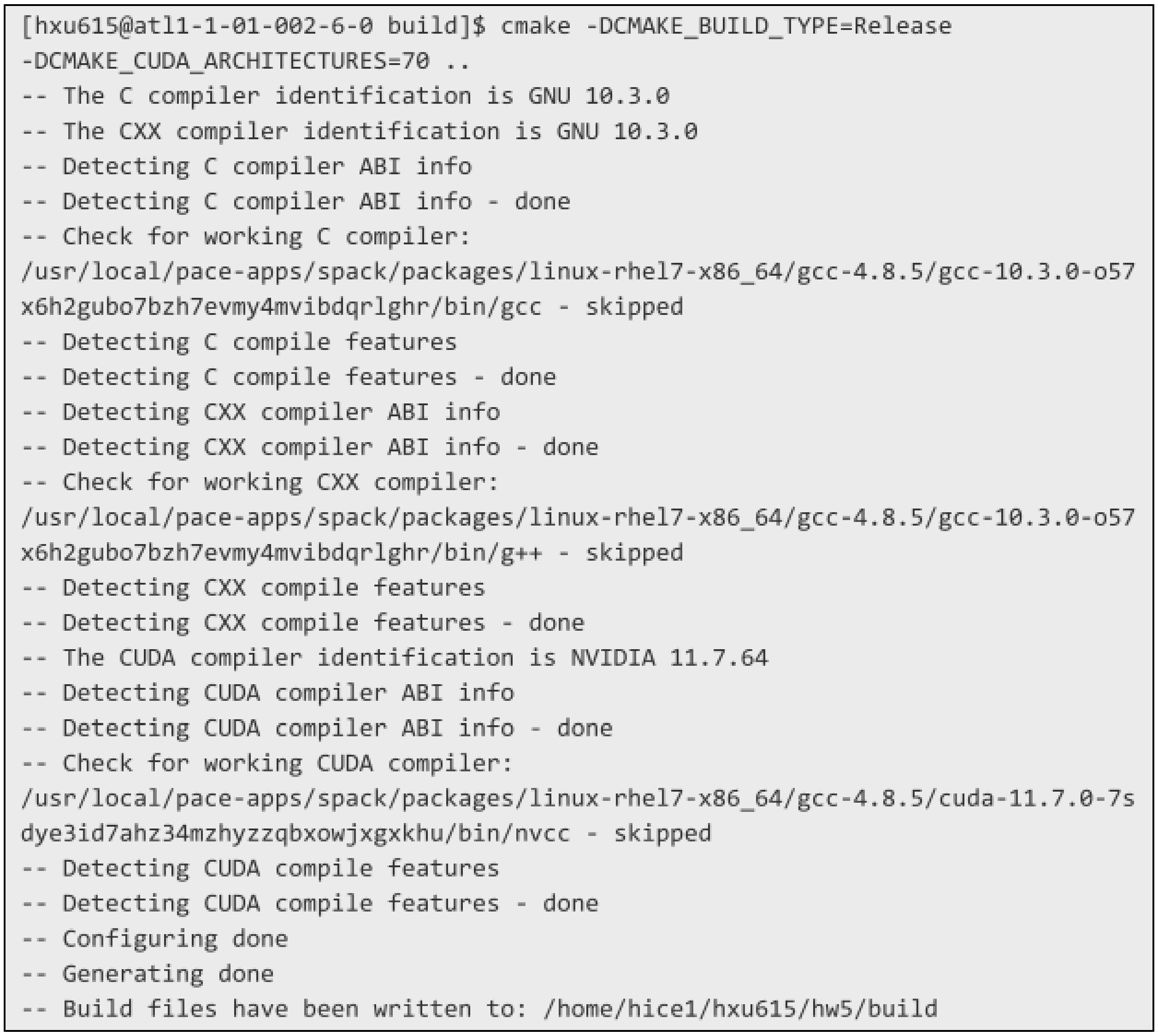
You can put these commands in your `~/.bash_profile` file to avoid typing them every time you log in.

Next, let's build the code. CMake prefers out of tree builds, so we start by creating a build directory.

In the hw5 directory:

```
$ mkdir build
$ cd build
```

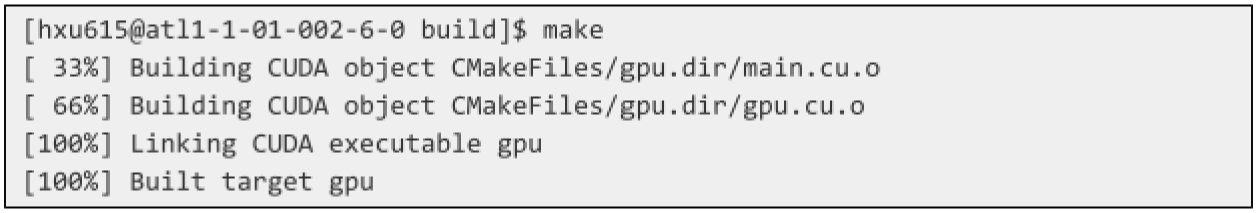
Next, we have to **configure our build.** We can either build our code in **Debug** mode or **Release** mode. For example:



```
[hxu615@atl1-1-01-002-6-0 build]$ cmake -DCMAKE_BUILD_TYPE=Release
-DCMAKE_CUDA_ARCHITECTURES=70 ..
-- The C compiler identification is GNU 10.3.0
-- The CXX compiler identification is GNU 10.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler:
/usr/local/pace-apps/spack/packages/linux-rhel7-x86_64/gcc-4.8.5/gcc-10.3.0-o57
x6h2gubo7bzh7evmy4mvibdqrlghr/bin/gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler:
/usr/local/pace-apps/spack/packages/linux-rhel7-x86_64/gcc-4.8.5/gcc-10.3.0-o57
x6h2gubo7bzh7evmy4mvibdqrlghr/bin/g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- The CUDA compiler identification is NVIDIA 11.7.64
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Check for working CUDA compiler:
/usr/local/pace-apps/spack/packages/linux-rhel7-x86_64/gcc-4.8.5/cuda-11.7.0-7s
dye3id7ahz34mzhyzzqbxowjxgxkhu/bin/nvcc - skipped
-- Detecting CUDA compile features
-- Detecting CUDA compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/hice1/hxu615/hw5/build
```

Be sure to include all the flags in CMake - i.e.,
`cmake -DCMAKE_BUILD_TYPE=Release` **`-DCMAKE_CUDA_ARCHITECTURES=70 ..`**

Once our build is configured, we may actually **execute** the build with **make**:

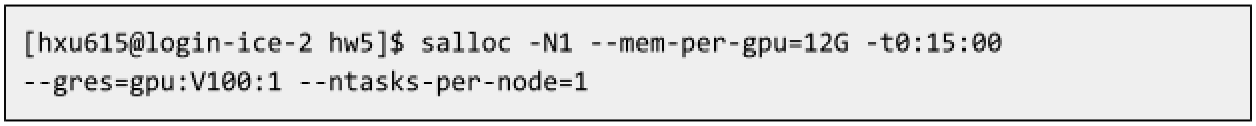


```
[hxu615@atl1-1-01-002-6-0 build]$ make
[ 33%] Building CUDA object CMakeFiles/gpu.dir/main.cu.o
[ 66%] Building CUDA object CMakeFiles/gpu.dir/gpu.cu.o
[100%] Linking CUDA executable gpu
[100%] Built target gpu
```

We now have a **binary** (gpu) and **job script** (job-gpu).

Running the code

You will need to test on a V100 GPU for this HW. Here is an example of how to get an interactive node with one:



```
[hxu615@login-ice-2 hw5]$ salloc -N1 --mem-per-gpu=12G -t0:15:00
--gres=gpu:V100:1 --ntasks-per-node=1
```

You can run the executable directly from the command prompt with:

```
$ ./gpu -n <num_particles>
```

Or you can use a job script - see job-gpu for an example.

You can check which GPU you got after the allocation with:

```
$ lspci | grep -i nvidia
```

or

```
$ nvidia-smi
```

You can also open another terminal and ssh into the allocated node.

For example, if I got `atl1-1-01-002-6-0` as my allocation, I would do:

```
$ ssh hxu615@atl1-1-01-002-6-0
```

then

```
$ watch -d -n .5 nvidia-smi
```

which will update the nvidia-smi regularly and show you the usage over time intervals.

Rendering output and checking correctness

Refer to the HW2 handout.

Workflow tips

Don't implement everything in GPU kernels all at once.

A generally useful approach:

- Start off implementing everything on CPU
- Move some part to a GPU kernel, and copy memory to and from GPU memory as necessary
- Eventually, everything is in a GPU kernel w/o any CPU to GPU memcpy
- For atomic operation, if atomicAdd/Sub/Inc/Dec/Min/Max are not sufficient, you can
- implement any atomic operations using atomicCAS (Compare And Swap).

You can find documentation about atomic functions in CUDA here.

Thrust is a C++ template library for CUDA based on the Standard Template Library.

`cuda-gdb` is a useful tool for debugging correctness.

`nvprof` is for performance debugging (memCpy time, kernel compute time).

Assignment

Implement the particle simulation in gpu.cu by filling in the `init_simulation` and `simulate_one_step` functions.

The code walkthrough and description can be found in the HW2 handout.

It may help to check the CUDA C++ guide.

Solution guide

Here is a guide from UCB CS267.

Grading and deliverables

This homework has two deliverables: the code and the writeup.

Code

We will grade your code by measuring the performance and scaling of your CUDA implementation.

Generating a parallel runtime plot

Please use the following sizes to generate a plot that shows how your parallel algorithm scales with problem size.

```
$ ./gpu -n 100000
$ ./gpu -n 200000
$ ./gpu -n 400000
$ ./gpu -n 800000
$ ./gpu -n 1600000
$ ./gpu -n 3200000
```

Writeup

Your write-up should contain:

- A plot in the log-log scale showing your CUDA code's performance and a description of the data structures that you used to achieve it.
- A description of the synchronization you used in the GPU implementation.

- A description of the design choices that you tried and how they affect the performance.
- A discussion of how your CUDA implementation compares in terms of performance to your OpenMP/MPI implementation (e.g., on the largest input size).

Point distribution

Correctness of your implementation is a precondition to get any marks.

As before, correctness is a prerequisite for performance points.

Point distribution:

Writeup - 1 point - includes
- Plot of the parallel CUDA runtime on one GPU, using the sizes above.
- Description of how you achieved the parallelization, design choices, data structures, and synchronization.
- Discussion of how the CUDA implementation compares to your other parallel (OpenMP, MPI) implementations.

3 points for raw performance (down is good)
- Deduction for raw performance = 3 - 3*(min(1, ta_time / student_time))
- Parallel times for ratio will be taken on the biggest input size specified in the parallel performance scaling plot in the handout.

TA_time = 44.2507 s

1 point for scaling of parallel linear-time implementation (lower slope is good)
- Deduction for parallel scaling = 1 - 1*(min(1, ta_slope / student_slope))
- Where parallel slope is calculated with x = input size, y = runtime using the input sizes in the handout.

TA_Slope = **1.094124**

That's the end of this homework! Submit your writeup and code as described in the "Homework submission instructions" below:

+

Homework submission instructions

There are two parts to the homework: a writeup and the code. The writeup should be in pdf form, and the code should be in zip form. **There will be two submission slots** in Gradescope called “Homework 4 - writeup” and “Homework 4 - code”. You should submit the writeup and code separately in their respective slots.

To get the code from PACE ICE to your local machine to submit, first zip it up on PACE ICE:

```
$ zip -r hw5.zip hw5/
```

Then from the terminal (or terminal equivalent) your local machine (e.g., your personal laptop), scp the code over.

```
$ scp <your-username-here>@login-ice.pace.gatech.edu:<path-to-hw5.zip>
<local-file-location>
```

For example, if I had `hw5.zip` in my home directory on PACE ICE and wanted to copy it into my current directory on my local machine, I would run:

```
$ scp hxu615@login-ice.pace.gatech.edu:~/hw5.zip .
```

Submission notes

Please follow the format above and name your code folder `hw5` and submit it in zipped form as `hw5.zip`.

Also, please limit your submission only to the source files and do not submit the `build/` folder.

Failure to follow instructions will result in penalties.

Test harness notes

You are free to edit the files besides gpu.cu to add extra printouts e.g. for debugging. For uniformity of outputs for the grading scripts, we will use the original test driver (main.cu) and common.h as in the handout. Please make sure your code (gpu.cu) works with the original handout test harness.