Assignment 4

Objectives

In this assignment, we will explore CUDA programming by optimizing two-dimensional array processing on GPUs.

Assignment

We will consider the same problem as that proposed in assignment 2. You will write a parallel program in CUDA that simulates heat transfer. The same restrictions on the two-dimensional array core and edges hold for this assignment. You will use CUDA to write an optimized simulation function targeting the GPUs on SCITAS cluster. Borrow ideas from assignment 2 and adapt them to the GPU architecture.

Development Infrastructure

On the course's Moodle page, we have provided the tarball A4.tgz, which contains the following files:

- `implementation.cu` – which contains `array_process` and `GPU_array_process`. Both functions have the same API: the first two inputs are pointers to the input and output arrays. Use one dimensional arrays as row-major two-dimensional arrays. The second input is the side length. The last input is the number of iterations for simulation.
  - `array_process` – which contains a CPU baseline implementation. Use this function as your baseline and reference.
  - `GPU_array_process` – implement the CUDA version for `array_process` in this function. The function passes the input array to the GPU, computes the simulation outcome, and then saves the final result in the output array. The function has code snippets to measure the time it takes to perform the three tasks. Include these snippets in your implementation to match the specified function outcome.
- `assignment4.cu` – the file containing the `main` function that will call `array_process` and/or `GPU_array_process`.
- `utility.h` – which contains a set of functions that the environment uses.

- `Makefile` – which compiles the environment using nvcc on the SCITAS cluster.
- `execute.sh` – A sample script to submit a job to the SCITAS cluster that compiles and runs the environment.
- `example.cu` – A simple CUDA example to help you develop your first CUDA program.

To use the environment:

1. Edit `execute.sh` to match your working environment and the input you want to execute.
2. Submit `execute.sh` using `sbatch` command on the cluster to compile your code and execute it.
3. The input arguments for `assignment4.cu` are, in this specified order:
    a. Side length
    b. Number of iterations
4. The program will save the array, after running `array_process` function, in `outputmatrix.csv`, and prints out all the measured time.

An example of running the program and the output it produces is shown below (numbers do not represent a realistic simulation):

```
# /bin/bash

$ ./assignment4 100000 500000

Host to Device MemCpy takes 17s

Computation takes 70s

Device to Host MemCpy takes 18s

Running the algorithm on 100000 by 100000 array for 500000 iteration
takes 100s

$ls

outputmatrix.csv ...
```

Implement a baseline GPU function first, insure its correctness, then optimize it.

Compare the output of your implementation with that of the baseline where every cell should be within a ±1 range of that of the baseline.

Make sure the printed times correspond to the functions they are performing.

Please note that it might be more practical for you to install CUDA on your machine (if it has a GPU) and start developing your code there before moving to the cluster to avoid the cluster's queueing time.

Deliverables

You need to turn in a tarball archive called `a4_<yourSCIPER>.tgz` to the Moodle link specified for Assignment 4, containing:

- An optimized version of `implementation.cu`.
- A report, with the name `a4_<yourSCIPER>.pdf`.

The tarball should not contain a directory. Create your tarball using this specific command:

```
$tar -czf a4_<yourSCIPER>.tgz /path/to/a4_<yourSCIPER>.pdf /path/to/implementation.cu
```

Report

In the report, we expect you to perform the following tasks:

1- Explain how your `GPU_array_process` implementation and its optimizations.
2- For each of the <length, iterations> combinations {<100,10>, <100,1000>, <1000,100>, <1000,10000>}:
    a. Report the results for running the baseline (reference CPU function), your optimized GPU version, and your optimized CPU version (OpenMP multithreaded version from assignment 2, running for 16 threads).
    b. Present graphically the breakdown of the GPU runtime for copying data from host to device, computation, and copying data from device to host.
    c. Analyze and explain the results of parts a and b.