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 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.

GPUs contain thousands of cores to accelerate graphics applications as well as scientific computation. However, to get the best out of the massive parallelism available in GPUs, you need to consider the characteristics of the underlying hardware architecture. In this assignment, you will implement a GPU kernel to parallelize the given program, and try various mapping and optimization techniques to maximize your speedup. We provide the single-thread CPU implementation, and your GPU implementation should produce exact output. We have reserved GPU machines for this assignment in SCITAS's izar cluster, which you can access with the command '*ssh username@izar.epfl.ch*'. You can run your Slurm scripts in this cluster in the same way as the previous assignments, but please make sure to use the script provided with the assignment as it contains extra arguments necessary for using the GPU machines.

Development Infrastructure

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

- `implementation.cu` – which contains two functions with the same API: `array_process` and `GPU_array_process`. In both functions, the first two arguments are pointers to the input and output arrays, the third argument is the side length, and the last argument 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` – this is the function you need to complete. It should contain a parallel and optimized CUDA version of `array_process`. The function passes the input array to 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.

- `assgignment4.cu` – the file containing the `main` function that will call `array_process` and/or `GPU_array_process`.
- `utility.h` – which contains a set of helper functions.
- `Makefile` – which compiles your code using `nvcc`.
- `execute.sh` – A sample script to submit jobs to the SCITAS cluster.

The environment will operate as follows:

1. The program file is called `assignment4.cu`.
2. Compiling the program using the provided `Makefile` produces the `assignment4` binary executable file.
3. The input arguments are, in this specified order:
    a. Side length
    b. Number of iterations (`N`)
4. The program will save the output, after running `N` iterations, in `outputmatrix.csv`, and prints out all the measured time.

Please note that it might be more practical for you to install CUDA on your machine (if it has an NVIDIA GPU) and start developing your code there before moving to the cluster to avoid the cluster's queueing 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 15s
Computation takes 70s
Device to Host MemCpy takes 15s
Running the algorithm on 100000 by 100000 array for 500000
iteration takes 100s
$ ls
outputmatrix.csv ...
```

Using the environment provided will guarantee that your program will comply with all specifications mentioned.

Deliverables

To submit your code and your report, create a tarball archive **(this is the only accepted format!)** called `a4_<yourGroupID>.tgz` and upload it on Moodle. Your file must contain:

- An optimized version of `implementation.cu`.
- A report, with the name `a4_<yourGroupID>.pdf` , that complies to the following description.

Your tarball should not contain a directory. Create your tarball using this specific command:
`$ tar -czf a4_< yourGroupID>.tgz /path/to/a4_<yourGroupID>.pdf /path/to/implementation.cu`

Report

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

1- Explain how you implemented and optimized `GPU_array_process`. List all the optimizations you applied and the reasons you chose to apply them. We are more interested in the optimizations than in parallelizing the program itself. Identify the different ways of splitting the work among threads and block of threads, and discuss their advantages and disadvantages.
2- Report how much each optimization improves performance and explain the result obtained. Design and run experiments that isolate the effect of each optimization/thread organization you tried out and then report the results.
3- For each following <length, iterations> combinations {<100,10>, <100,1000>, <1000,100>, <1000,10000>}:
    a. In a graph, present the execution time you measured for running the baseline CPU version and your optimized GPU version.
    b. Present graphically the breakdown of the GPU runtime for copying data from host to device, computation, and copying data from device to host.
4- Analyze and explain your results from the previous part.

The report should be as precise as possible, addressing the four questions above. Keep the descriptions of the algorithm implemented and of your reasoning in parallelizing the program short. A regular report should not be much longer than 3 pages.

Grading

The code will be graded automatically by a script. The script will check the running time of your program for different configurations and if the results returned are consistent with what was expected. Plagiarized code will receive 0. The reports will be read and graded by humans. The grade breakdown is as follows:

- Correctness: 50%
- Report: 50%