Lab 5: BFS Queueing

EE5355 - Parallel Algorithmic Techniques

Due: See Moodle

1 Objective

The purpose of this lab is to understand hierarchical queuing in the context of the breadth first search algorithm as an example. You will implement a single iteration of breadth first search that takes a set of nodes in the current level (also called wave-front) as input and outputs the set of nodes belonging to the next level.

2 Procedure

Step 1: Transfer the tarfile lab5.tgz into your project directory and extract it to create the lab5 directory.

Step 2: In the file `kernel.cu`, you will find a function named `cpu_queuing`. This function implements a simple CPU version of a single iteration of BFS with queuing.

Compile and test the code:

```
make
./bfs-queuing <m>          # Mode: m, Nodes: 200,000, Max neighbors per node: 10
./bfs-queuing <m> <N>      # Mode: m, Nodes:       N, Max neighbors per node: 10
./bfs-queuing <m> <N> <M># Mode: m, Nodes:       N, Max neighbors per node: M
```

For the mode option, select mode 1. There are 4 modes available for this lab:

- Mode 1 executes the CPU version
- Mode 2 executes the GPU version using just a global queue
- Mode 3 executes the GPU version with hierarchical queuing with block and global queuing
- Mode 4 executes the GPU version with hierarchical queuing with warp, block, and global queuing

Mode 1 is provided to you and you will be implementing modes 2-4 throughout the lab.

Step 3: Edit the kernel `gpu_global_queuing_kernel` in the file `kernel.cu` to implement the algorithm using just a global queue. Compile and test the code using mode 2.

Step 4: Edit the kernel `gpu_block_queuing_kernel` in the file `kernel.cu` to implement the algorithm using block and global queuing. Compile and test the code using mode 3.

Step 5: Edit the kernel `gpu_warp_queuing_kernel` in the file `kernel.cu` to implement the algorithm using warp, block, and global queuing. Compile and test the code using mode 4.

Step 6: Edit the file `report.txt` to answer the following questions:

1. Try running all four implementations using varying input sizes. Compare the performance and scalability of each implementation. Explain why you think each implementation performs better or worse than the one before it.

Step 7: Submit your assignment. You should only submit the following files:

- `kernel.cu`
- `report.txt`

3 Grading

Your submission will be graded based on the following criteria.

- Functionality/knowledge: 75%
  - Correct code and output results for mode 2
  - Correct code and output results for mode 3
  - Correct code and output results for mode 4
  - Performance
- Answers to question: 25%
  - Thoroughness of performance comparison and explanations demonstrating a clear understanding of the factors influencing performance
  - Neatness and clarity