Lab 1: Scatter-to-Gather Transformation

EE5355 - Parallel Algorithmic Techniques

Due: See Moodle

1 Objective

The purpose of this lab is to understand scatter-to-gather transformation in the context of a simple computation example.

2 Procedure

Step 1: Transfer the tarfile lab1.tgz into your project directory and extract it to create the lab1 directory.

Step 2: In the file `kernel.cu`, you will find a function named `s2g_cpu_scatter`. This function implements a simple scatter pattern on CPU. It loops over an input array, then for each input element it performs some computation (*`outInvariant(...)`*), loops over the output array, does some more computation (*`outDependent(...)`*), and accumulates to the output element.

Compile and test the code:

```
make
./s2g <m>              # Mode: m, Input: 4,096, Output: 4,096
./s2g <m> <M>          # Mode: m, Input:     M, Output:     M
./s2g <m> <M> <N>      # Mode: m, Input:     M, Output:     N
```

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

- Mode 1 executes the CPU scatter version
- Mode 2 executes the CPU gather version
- Mode 3 executes the GPU scatter version
- Mode 4 executes the GPU gather version

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

Step 3: Edit the function `s2g_cpu_gather` in the file `kernel.cu` to implement a gather version of `s2g_cpu_scatter` on the CPU. Compile and test the

code using mode 2.

Step 4: Edit the kernel `s2g_gpu_scatter_kernel` in the file `kernel.cu` to implement a scatter version of `s2g_cpu_scatter` on the GPU, and edit the function `s2g_gpu_scatter` to launch the kernel you implemented. Compile and test the code using mode 3.

Step 5: Edit the kernel `s2g_gpu_gather_kernel` in the file `kernel.cu` to implement a gather version of `s2g_cpu_scatter` on the GPU and edit the function `s2g_gpu_gather` to launch the kernel you implemented. Compile and test the code using mode 4.

Step 6: In a new file named report.txt, answer the following questions:

- Try running all four implementations using varying input combinations. Use at least 5 different combinations. Try scaling the input and output sizes differently, having both low, both high, one low and one high, etc. Comment on the relative behavior of each implementation. Explain why you think certain implementations perform better than others for various input combinations.

Step 7: Submit your assignment. You should only submit the following files as a tarfile via Moodle.

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

```
tar -zcvf lab1_submission.tgz 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
  - Correct usage of atomic operations and other performance enhancing optimizations where possible
  - Performance
- Report: 25%
  - Thoughtfulness of input combinations used

- Thoroughness of performance comparison and explanations demonstrating a clear understanding of the factors influencing performance
- Neatness and clarity