cb4904, Caraline Bruzinski

HPC
Assignment 4

April 23, 2023

1. Greene Network Test.

I added an appropriate sbatch file to the homework repository with instruction to run 1 cpu per node across 2 nodes. To ensure I was really running across 2 nodes of the network, I tested the process by adding sample code to pingpong which prints the host name as we used in class. My code is available in the zip file. Generally, it seems like I am assigned 2 nodes which are next to one another. Example output for the network test is below:

```
Rank 0/0 running on gv001.hpc.nyu.edu.
Rank 1/0 running on gv002.hpc.nyu.edu.
pingpong latency: 1.284747e-03 ms
Rank 0/0 running on gv001.hpc.nyu.edu.
Rank 1/0 running on gv002.hpc.nyu.edu.
pingpong bandwidth: 1.135798e+01 GB/s
```

Sometimes I am assigned different nodes on the next run (nodes in [gv001,gv002], [gr003,gr004]). With these results I can be sure that the code runs across the network on two different nodes. However, I did not want the execution of printing the hostname to affect any latency or bandwidth measurement, so I commented these lines out of pingpong and the result is:

```
pingpong latency: 1.890761e-03 ms
pingpong bandwidth: 1.091617e+01 GB/s
```

2. MPI Ring Communication.

My two code files are provided in the repository under 'int_ring.cpp' and 'int_ring_bandwidth.cpp'. For testing the latency I ran with several values for N repeats. My timings for the procedure across the three nodes is provided in the table below:

Latency, varying N Repeats

N	Latency (ms)	Nodes Assigned
100	7.338e-03	ga[010-012]
1,000	3.119e-03	cs[294-296]
10,000	3.086e-03	cs[294-296]
100,000	3.054e-03	cs[294-296]
1,000,000	3.388e-03	cs[294-296]
10,000,000	3.227e-03	cs[294-296]

I thought it was strange that the first run has the highest latency, so I output the nodes assigned to me for each run. No matter how many times I ran with N=100 I was still assigned compute nodes ga[010-012], while I was consistently assigned nodes cs[294-296] otherwise. In Ed Stem, professor said that all of the compute nodes are the same, but maybe it's possible that the way they are configured in the network is different. In my case, perhaps SLURM has an algorithm for assigning the workloads and decided that my code with N=100 repeats should run on ga[xxx] nodes for whatever reason, while assigning other workloads to other machines.
Focusing on the runtimes for N >100, the latency remains generally consistent across all runs, no matter as N grows. We can deduce that the latency we see in the results is likely close to the latency of the actual network, especially with the .
For testing the bandwidth of the network, I changed the integer message to be a 2MB array. My results for measuring bandwidth of the network are in the table below:

Bandwidth, varying N Repeats

N	Bandwidth (GB/s)	Nodes Assigned
100	4.045e+03	cs[493-495]
1,000	4.186e+03	cs[493-495]
10,000	4.199e+03	cs[493-495]
100,000	4.202e+03	cs[493-495]
1,000,000	7.317e+03	ga[010-012]
10,000,000	DNF	DNF

Here I notice similar behavior between the assignment of nodes in the network, and the achieved bandwidth of the network. My first few runs are assigned cs[493-495] and attain a similar bandwidth result (4,000 GB/s). When N is large my program was assigned different nodes, and had a higher bandwidth of 7,000 GB/s. I imagine that the SLURM scheduler decided to use particular configuration for a given workload. Regardless, we can say that given the consistent result when assigned cs[xxx] nodes, that the attainable bandwidth of the network is roughly 4,000 GB/s.
Another thing to note, is that my final run with N=10M did not finish. I tried to play with a few different memory settings in the slurm job file, but I still kept having a backtrace with memory errors. I'm not sure if I needed to change a different setting in the configuration file in order to get this run to work through the network.

3. Implement an MPI version of the scan function we implemented on OpenMP.

For this question I chose to implement the MPI version of the scan procedure we implemented in the previous homework. In order to get a comparison of the timings for how this type of job is affected by the network, I also imported the sequential scan version and performed this on the root node (rank 0). My results are in the table below:

Timings for varying compute nodes, size N=100

Nodes	Sequential Scan (ms)	MPI Scan (ms)
3	0.002	0.278
4	0.002	8.279
5	0.002	0.197

Sequential scan is the same for each number of nodes. I only run the sequential scan on the root node and it doesn't have to go through the network. In this spirit it seems intuitive that the time taken by sequential scan is faster than the time taken by MPI scan. Surprisingly I got the best performance in my results when I used 5 nodes. Originally I thought that the less nodes I used the better my performance would be, as there would be less hops through the network.

Overall, I thought that I would be able to see consistent trends in the behavior based on the number of nodes used, but each time I ran the code for a given number of processors I got slightly different results. I think that communication through the network is not always consistent, or it might depend if the nodes are placed near each other on the network. Some runs SLURM would provide me consecutive nodes, other runs I would get nodes starting with a different prefix. It also might depend on what other traffic is being communicated through the network during my runs.

4. Pitch your Final Project.

Introduction
We aim to speed up a sequential, bare bones version of the Burning Ship set computation. Burning Ship is a fractal which is not perfectly self-similar. It can be computed with an escape time algorithm, which imposes a specific set of constraints that each coordinate on the complex plane must satisfy. The number of iterations taken before constraints are satisfied, determines whether or not the coordinate point is in the fractal set or not. When we view the result, we are really observing the number of iterations is took for a coordinate to meet specific criteria. The grayscale value corresponds to the number of iterations taken by each coordinate point.

Background: Fractals become complex at their border regions
Some fractals take interesting form around the border regions of its primary shapes, where changing the coarseness (the decimal precision) of the coordinate by a very small number makes a difference of whether the coordinate point will be included in the set or not (ie. 1.2137 is in the set, but 1.21372 is not). In this way, border regions have a high alternating frequency of (not in set) and (in set) values. This leads to beautiful designs as we increase the decimal precision and corresponding image size.

Objective
Our goal is to speed up the escape-time algorithm which determines whether a coordinate lies in the set or not. The current code sequentially iterates over the grid of pixels, maps each to a coordinate in the complex plane, and then starts the iteration for that point.
We propose a new approach, which evaluates the impact of parallelizing the code with OpenMP, and uses SIMD instructions to get better performance. Additionally, we will explore the use of CUDA to see if we can leverage GPU nodes which are well suited for this type of simple iteration work.

Work Plan
I will be working with KyuDoun Sim (ks6401) on this project, and we will be comparing results from the following methods:

1. Baseline (sequential) version of the escape time algorithm
2. OpenMP version
3. OpenMP with SIMD
4. CUDA version