Naman Gangwani
nkg160030

CS 4301.001
May 1, 2018

Report for Pgm5

Problem Statement

Sorting billions of numbers can be quite a large and costly task, especially when dealing with 32-bit numbers. Making use of multiple processors and cores to divide the tasks can help in improving its performance. One way to specifically do this is by utilizing MPI to split work amongst multiple processes and the OpenMP API for C. MPI functions and OpenMP directives will be useful in improving a quicksort algorithm for sorting a large series of numbers that ultimately comprise of the Parallel Sorting by Regular Sampling (PSRS) algorithm through the process of parallelization.

Approach to Solution

MPI is the main tool utilized to assist in finding a solution to the problem, although OpenMP also helps in parallelizing critical sections, parallelizing loops, privatizing variables in nested loops, and providing reduction clauses. Additionally, SLURM (Simple Linux Utility for Resource Management) and access to a Linux server on a supercomputer through an SSH may be essential. Since it is apparent that sorting 32 series of 128,000,000 numbers is quite resource intensive, the decision to split up the work amongst multiple processes and partitioning into segments to pass to their corresponding processes through MPI functions was made. Similarly, OpenMP pragma directives for critical sections and loops were utilized in order to parallelize the pool of recursion in the quicksort algorithm and the loops that index through the elements of the node's assigned numbers. Combined together in the C programming language, MPI and OpenMP make a powerful team in implementing the PSRS algorithm.

Solution Description

In this task, the code written for the matrix-vector multiplication was written completely with personal knowledge of multiplying a matrix with a vector and then wrapped with MPI functions and OpenMP directives. The SLURM file from Pgm4 was reused with extra modifications made by increasing the execution time limit and the number of processes.

When logged into a remotely hosted Linux server on a supercomputer through an SSH (with bridges.psc.edu being the recommended remote host), it is important to have the *Pgm5.c* and *Pgm5.sh* files in the current directory. When they are there, go into interact mode by typing in "interact" and pressing enter. From there, executing the bash script with the SLURM-specific command *sbatch* should be done with the command *sbatch Pgm5.sh*. A slurm file should be produced in the current directory from executing this bash script, and the content of that file will display the size of the problem along with the statistics of the program that ran, such as the elapsed time when sorting, and the samples per second.

```
[namang@br018 ~]$ interact

A command prompt will appear when your session begins
"Ctrl+d" or "exit" will end your session

[namang@r002 ~]$ sbatch Pgm5.sh
Submitted batch job 2910046
[namang@r002 ~]$ cat slurm-2910046.out
Process 2 initial: 274 380 120 300 341 365 480 93
Process 0 initial: 383 293 415 386 386 335 492 277
Process 1 initial: 459 77 429 70 443 160 303 257
Process 0 final: 70 77 93 120 160 257 274 277 293 300
Process 1 final: 303 335 341 365 380 383 386 386
Process 2 final: 415 429 443 459 480 492

Elapsed time for PSRS algorithm to complete: 0.87 seconds
Samples per second: 27.58
```

The execution of the bash script and the results after sorting randomly generated numbers amongst 3 processes and 32 nodes.

For the purpose of verifying that the results were correct, the program was ran with only 3 processes with 8 randomly generated numbers per process. This allowed the values to be printed cleanly. However, the amount can be adjusted to 128,000,000 by changing the N value in the program. Seeing that the sorting worked properly, it is worthy to note the elapsed time was less than one second and the samples per second were greater than the total number of samples, indicating a good use of time. 0.87 seconds can be considered slow for 24 samples, but this is a small problem size. As the problem size gets bigger, there should be more appropriate elapsed times. Although the Bridges SLURM queue was too busy to allocate enough resources for 128,000,000 elements, it is apparent that this PSRS algorithm would have executed at dozens of millions of operations per second. When ran sequentially, the program took approximately 1.35 seconds to complete. Thus, the speedup is 1.35/0.87, or 1.55. When plugging this speedup value into the formula for the Karp-Flatt metric along with 3 for p (since 3 processes were specified in the SLURM file when running the program), we retrieve a value of 0.56 for the Karp-Flatt metric, which (while considering a small problem size for verification purposes), ensures success in the parallelization of the PSRS algorithm for sorting samples amongst multiple processes due to parallel overhead throughout the MPI sending/receiving and other sources of overhead such as process startup time and synchronization time being accounted for.