Naman Gangwani CS 4301.001
nkg160030 April 8, 2018

Report for Pgm4

Problem Statement

Multiplying a matrix with considerably large dimensions with a vector can consequently be a large task, especially with matrix that end up containing millions or billions of elements. Making use of multiple processes and cores 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 sequential algorithm for matrix-vector multiplication 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 does help in 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 computing a vector by multiplying a 32768x16384 matrix with a 16384x1 vector is quite resource intensive, the decision to split up the work amongst multiple processes and combining the solution from each process into the final vector through MPI functions was made. Similarly, an OpenMP pragma directive was utilized in order to parallelize the loops that index through the elements of the process's assigned rows and compute the dot product with the vector it is being multiplied by. Combined together, MPI and OpenMP make a powerful team in matrix-vector multiplication.

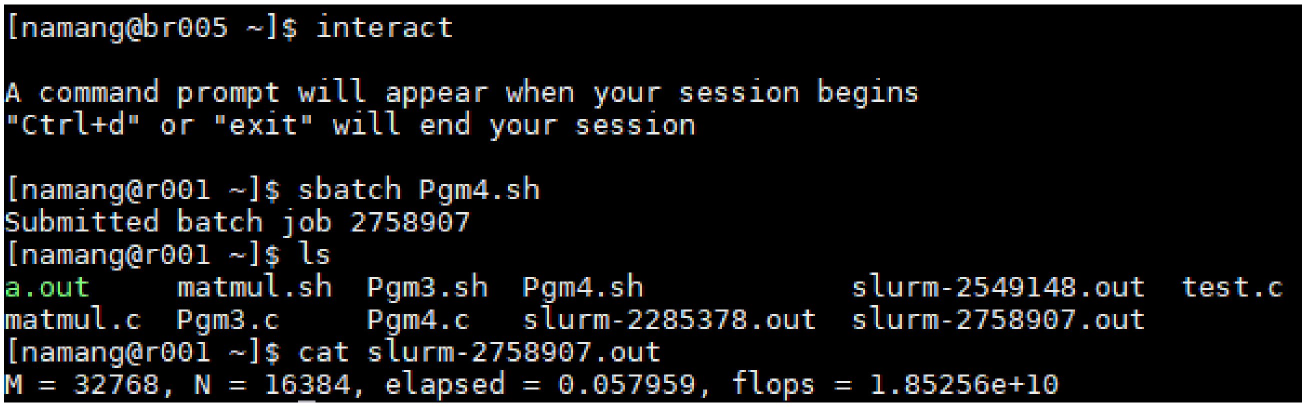
Solution Description

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 Pgm3 was reused with an extra modification flag (*-mcmodel=large*) to not limit memory, and the number of processes was increased.

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 *Pgm4.c* and *Pgm4.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 Pgm4.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 dimensions of the matrix along with the statistics of the program that ran, such as the elapsed time and the floating point operations per second.

Naman Gangwani
nkg160030

CS 4301.001
April 8, 2018



```
[namang@br005 ~]$ interact

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

[namang@r001 ~]$ sbatch Pgm4.sh
Submitted batch job 2758907
[namang@r001 ~]$ ls
a.out     matmul.sh  Pgm3.sh  Pgm4.sh            slurm-2549148.out  test.c
matmul.c  Pgm3.c     Pgm4.c   slurm-2285378.out  slurm-2758907.out
[namang@r001 ~]$ cat slurm-2758907.out
M = 32768, N = 16384, elapsed = 0.057959, flops = 1.85256e+10
```

The execution of the bash script and the results after multiplying the 32768x16384 matrix with the 16384x1 vector.

When running this program, it is apparent that the process of parallelization through MPI functions and OpenMP directives was quite effective, for it completed its calculation of the matrix-vector multiplication of sample matrix and vector (in the code) in approximately 0.06 seconds. When ran sequentially, the program took approximately 0.48402 seconds to complete. Thus, the speedup is 0.48402/0.057959, or 8.351. When plugging this speedup value into the formula for the Karp-Flatt metric along with 32 for p (since 32 processes were specified in the SLURM file when running the program), we retrieve a value of 0.09135 for the Karp-Flatt metric, which ensures success in the parallelization of the matrix-vector multiplication algorithm 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, because this Karp-Flatt metric value was steadily increasing as the speedup and number of processors increased.