Report for mpi4py lab

This is a report of the lab of matrix multiplication using mpi4py.

Discussion

Benchmarking Results

First, I benchmarked the matrix multiplication using *mpi4py* in parallel on multiple processors and in serial on on one processor. The benchmark results show that my parallel code is correct, which is the same as the the serial code. In comparing the results of the serial code and the parallel code, I use the following command:

```
assert np.allclose(C_parallel, C_serial), "MPI result != Serial result"
```

Speedup/Slowdown Analysis

The table below shows the running time for different number of processors and different sizes of the matrix, based on the timestamps of the output log files.

Data Type	Total Processors	Node	Matrix Size	Time (sec)
np.float32	16	2	20240	27
np.float32	64	4	20240	58
np.float32	16	2	40960	149
np.float32	64	4	40960	131
np.float64	16	2	20240	46
np.float64	64	4	20240	64
np.float64	16	2	40960	291
np.float64	64	4	40960	192

1. For different data types, the matrix multiplication on the np.float32 data type is faster than the np.float64 data type in general. This is because the np.float32 data type is more efficient than the np.float64 data type in terms of memory usage and computation.
2. For different matrix sizes, the matrix multiplication on the larger matrix size takes longer time to finish the computation. This is because the larger matrix size requires more memory and more computation.
3. For different number of nodes, the matrix multiplication on the larger number of nodes takes less time to finish the computation for larger matrix size. However, for smaller matrix size, the matrix multiplication on the larger number of nodes takes more time to finish the computation. This is because the smaller matrix size does not require more memory and computation, and the communication overhead is larger than the

computation time. Also, processors may be different on different nodes as I submitted the job to the cluster on random nodes that are available.

Content of the repository

Code and Report

1. *README.md*: this file
2. *Report.pdf*: PDF version of this file
3. *matmult_mpi_float32.py*: the main code for matrix multiplication using mpi4py on np.float32 data type
4. *matmult_mpi_float64.py*: the main code for matrix multiplication using mpi4py on np.float64 data type
5. *matmult_mpi_benchmark_serial_float32.py*: the benchmarking code for matrix multiplication using mpi4py on np.float32 data type
6. *matmult_mpi_benchmark_serial_float64.py*: the benchmarking code for matrix multiplication using mpi4py on np.float64 data type

Slurm Script

1. job_benchmark_float32.slurm
2. job_benchmark_float64.slurm
3. job_size20240_proc16_float32.slurm
4. job_size20240_proc16_float64.slurm
5. job_size20240_proc64_float32.slurm
6. job_size20240_proc64_float64.slurm
7. job_size40960_proc16_float32.slurm
8. job_size40960_proc16_float64.slurm
9. job_size40960_proc64_float32.slurm
10. job_size40960_proc64_float64.slurm

I also provided the automated script to submit all the slurm scripts to the cluster:

```
bash submit_job.sh
```

Output Log Files

1. **./outputs/job_benchmark_float32.sh03-05n02.12436604.out**: the output log file for the benchmarking code on np.float32 data type
2. **./outputs/job_benchmark_float64.sh03-05n02.12436604.out**: the output log file for the benchmarking code on np.float64 data type
3. **./outputs/job_size20240_proc16_float32.sh03-05n02.12436604.out**: the output log file for the main code on np.float32 data type with matrix size 20240 and 16 processors

4. ... and so on

Note

1. The benchmarking code is used to test the accurate of the matrix multiplication using mpi4py in parallel on multiple processors and in serial on on one node.
2. The main code is used to test the performance of the matrix multiplication.
3. The slurm script is used to submit the job to the cluster and the automated script is used to submit all the slurm scripts to the cluster.
4. All the codes are tested on the Sherlock cluster (**serc** partition) at Stanford University, instead of the Google Cloud Platform, and the later responds too slow to me due to unknown reasons. The following python is used on the Sherlock:

```
module load  py-mpi4py/3.1.3_py39
```