Homework #5 Parallel Matrix-Vector Multiple with Performance Analysis

Overview

You are going to write and evaluate a parallel implementation of matrix-vector multiplication. You may implement any of the three algorithms we talked about in class. For this assignment, you will read and write files using HDF5. At a minimum, you must also use the following MPI functions in your implementation: `MPI_Bcast`, `MPI_Scatter`, and `MPI_Gather`.

Your written evaluation of the algorithm is half of your grade. You will run a series of tests and explore some of the concepts we have discussed including Amdahl's law, Amdahl's Effect, the Karp-Flatt metric, speedup and efficiency.

Software Development

Write a program called `multiply` that multiplies a matrix and a vector. Your program should accept five inputs:

- Matrix input filename: specified as `-m <filename>`
- Vector input filename: specified as `-v <filename>`
- Output filename (optional): if specified, use `-o <filename>`
- Verbose (optional, see discussion below): if used, specify `-verbose`
- Partial read (optional, see discussion below: used, specify `-partial`

Examples:

```
mpirun -np 4 ./multiply -m m1000.hdf5 -v v1000.hdf5 -partial
mpirun -np 8 ./multiply -m m1000.hdf5 -v v1000.hdf5 -verbose
mpirun -np 4 ./multiply -m m1000.hdf5 -v v1000.hdf5 -o out
./multiply -m m1000.hdf5 -v v1000.hdf5
```

You may assume your input matrices will be square and that the size will always be evenly divisible by number of processors. Your code must also run in serial.

You should use the example HDF5 code provided on the d2L site to assist you with reading and writing HDF5 files. You will need to have two methods for reading in the matrix an distributing it to the ranks (if running in parallel):

- Full read and distribute: This is the default behavior. One of the ranks will read the entire file and distribute it to the other ranks using the `MPI_Scatter` method.
- Partial read: Here, one of the ranks will read only the information from the file necessary to distribute to a single rank. One this rank has received it's information, the process continues until all ranks have their portion of the matrix.

Once each rank has computed it's portion of the answer, you should collect the solutions using `MPI_Gather`.

You can use the code provided to generate matrix and vector files in HDF5. Feel free to modify this code in anyway you want to make your performance better.

Only output your data if the -o <filename> flag is specified. Output is is a problem size x 1 HDF5 matrix 2D matrix (see helper code).

The verbose mode should collect timing information. Please output the number of processors used, the problem size, the total time, the time spent in serial execution, the time spent performing IO, the parallel processing time, and the communication time. This is not trivial when different processors are performing different tasks. You many need to collect and aggregate some of the metrics. You may want to warm-up the `MPI_Wtime`.

Analysis

You will be running a series of experiments. Consider writing bash scripts (or your favorite scripting language) for managing the execution of the different trials.

I. Serial analysis and Amdahl's Law

Profile your serial implementation and estimate the percentage of time your program spends doing serial work (everything but the multiplication). Do this for problems of size 1K, 5K, 10K, 20K, and 40K. Based on this information and **Amdahl's Law**, how do you anticipate your algorithm will scale as you increase the number of processors for a give problem size? Remember, Amdahl's law doesn't require you to account for communication or additional MPI overhead. Show your data and explain your calculations.

II. Parallel analysis: variable problem size

Run your parallel implementation for problems of size 1K, 10K, 20K, and 40K using using 8 procesors. Do this for both the partial and full read for a total of 8 individual experiments. Repeat each experiment 10 times and average your output.

1. Plot the communication time and IO time as a function of problem size.
2. Which method is more efficient: partial read with MPI_Send or full read with MPI_Scatter in terms of IO and communication? Does the size impact your answer? Please site your data when justifying your answer.

III. Parallel analysis: variable number of processors

Run your experiments using 1, 2, 4, and 8 processors using problem sizes 1K, 10K, 20K, and 40K. Do this experiment for only a single IO implementation (your choice). Repeat these experiments 10 times and take the average.

1. Plot the **speedup** for each problem size as a function of the number of processors.
2. Plot the **efficiency** for each problem size as a function of the number of processors.
3. How does the speedup change with problem size?
4. Do you notice **Amdahl's effect** for this problem? Why or why not?
5. **Karp-Flatt Metric**: Compute the Karp-Flatt serial fraction metric for the 20K problem size as a function of the number of processors. What does this metric suggest is the primary reason for the speedup you are witnessing? Please show your data and explain your answers.

Assignment Submission

Your lab write up, answers to questions, plots and all other bits should be presented using formal scientific write-up guidelines including, as a minimum, the following sections; Introduction, Methods, Results, and Conclusions. If you use outside information make sure you properly reference them. Make sure all graphs are properly scaled, labeled, titled and referenced.

Create a file named username-hw5.tar.gz that contains a softcopy of your report, your source code and makefiles. Please do not include object files, executables, or any *matrix files*. Submit the tar file to the d2L site by the due date. Bring a printed copy of your report to lab. The assignment is due *before lab* on Friday, March 3rd.