CSE 661 – Parallel and Vector Architectures

Parallel Programming Assignment

Due Wednesday, October 31, 2007 by 12 Midnight

Objectives:

- Developing, writing, testing, and debugging parallel programs
- Using the message passing interface library
- Parallel program performance evaluation
- Teamwork

Problem Statement

Gaussian elimination is a well-known technique for solving simultaneous linear systems of equations. Variables are eliminated one by one until there is only one left, and the discovered values of variables are back-substituted to obtain the values of other variables. In practice, the linear equations are represented as an augmented matrix $A[n][n+1]$ with n rows and $n+1$ columns. The matrix is converted to an upper triangular matrix. Then back substitutions are used.

Pseudo-code for sequential Gaussian elimination is shown below. The diagonal element for a particular iteration of the *k* loop is called the *pivot elemen*t, and its row is called the *pivot row*.

```
procedure GaussianElimination (A[n][n+1]) {
   for k = 0 to n-2 do {
      for i = k+1 to n-1 do {
         factor = A[i][k]/A[k][k];
         for j = k+1 to n do {
            A[i][j] = A[i][j] – factor * A[k][j];
         }
         A[i][k] = 0
      }
   }
}
```

1. Draw a simple figure illustrating the dependences among matrix elements. The input matrix should be read from a text file and the output matrix should also be produced in a text file. Large $n \times n$ matrices can be generated randomly. Write a small program to generate $n \times n$ matrices. Generate 1000×1001 to 10000×10001 element matrices.

2. The pivot element and the pivot row can be effectively broadcast directly to all processes that need it. This is called the *broadcast version*. Parallelism is exploited only within an iteration of the outermost loop. Assuming a decomposition into rows, write an MPI parallel version of the Gaussian elimination procedure.

3. Gaussian elimination can also be parallelized in a form that is more aggressive in exploiting the available concurrency, even across outer loop iterations. During the *k*th iteration, the process assigned the pivot row can simply pass the pivot row on to the next process instead of broadcasting it. This process can use the pivot row to update its assigned rows immediately, as well as pass it on to the next process, and so on. As soon as this process has done its computation for the *k*th iteration of that loop in the sequential program, it can immediately perform its pivot row computation for the (*k*+1)th iteration without waiting for all other processes to receive the *k*th row and perform their work for the *k*th iteration. It can then pass this (*k*+1)th row on to the next process as well, which can use it right away instead of waiting for the entire previous *k* loop iterations to complete. Multiple *k* loop iterations are in progress at once. This is called the *pipelined* form of parallelism. Write a message-passing parallel version of the Gaussian elimination procedure using the *pipelined* form of parallelism.

Tools:

Your program should be developed and tested on the linux cluster machines 1 through 20, using the installed version of MPICH.

Groups:

Two students can form a group. Make sure to write the names of the students involved in your group on the project report.

Coding and Documentation:

Develop the code for the two given problems with the following aspects in mind:

- Correctness: the code should work properly.
- Efficiency: proper partitioning of data, efficient communication, overlap of computation with communication.
- Documentation: the code should be well documented through the appropriate use of comments. Use a proper standard coding style.

Report Document:

The project report must contain sections highlighting the following:

- **Program Design**

  Two parallel programs should be developed: first one will use the broadcast approach and the second will use the pipelined approach. Specify clearly the design of the Gaussian Elimination procedure for each version. Specify the decomposition and assignment of the data to the various processes. Indicate the communication requirement between the various processes. Discuss the advantages/disadvantages of each parallel version.

- **Program Output and Performance**

  Run both programs using different input files and on a different number of machines. At least, three matrix sizes should be tested: 1000×1001, 2000×2001, and 4000×4001 elements. You may also test larger matrices if the execution time is reasonable. Repeat the execution with different number of processes. Try 1, 2, 4, 8, 16, and 32 processes. If

the number of processes exceeds the number of machines then some machines will end up having more than one process.

Verify the correctness of the output of both parallel programs. The best way to do it is to write a sequential version of the program that generates the output in a file. Compare the output of the sequential program against the output of the parallel versions of the program.

For each experiment, specify the input matrix size and the number of processes. Compute and record the Wall execution time. Collect data in tables and draw charts showing the execution time versus the number of processes and the input size. Also compute and show the speedup of computation.

- **Discussion**

  Discuss all the various inputs and configurations that were handled. Discuss the speedup of the computation as the number of processes increases. You may go beyond 32 processes if you wish. Also discuss the effect of increasing the input size on the execution time.

  Compare the two parallel program versions and discuss their efficiencies and their communication overheads.

  Compare the execution time of the sequential version of the program against the parallel versions to find the speedup and the overhead of parallel execution. Discuss the difference in the execution time when you run the sequential version and the parallel versions on a single machine.

- **Teamwork**

  Group members are required to divide the work equally among themselves, so that everyone is involved in program development, and debugging.

  Show clearly the division of work among the group members using a Chart and also prepare a Project execution plan showing the time frame for completing the subtasks of the project.

Submission Guidelines:

All submissions will be done through WebCT.

Submit one zip file containing the source and binary code of all programs as well as the report document. Make sure that all programs are well documented.

Grading Policy:

The grade will be divided according to the following components:

- Correctness of code: program produces correct results
- Efficiency of code: achieving speedup in the parallel versions
- Documentation of code: program is well documented
- Team Work: participation and contribution to the project
- Report document: report is well written and results are well reported and discussed

Late Policy:

The project should be submitted on the due date by midnight. Late projects are accepted, but will be penalized 5% for each late day and for a maximum of 5 late days (or 25%). Projects submitted after 5 late days will not be accepted.