Riten Chhatrala
rchhatrala@hawk.iit.edu
CWID:-A20385102

Assignment-2

Q-1. In this problem you will analyze the performance of distributed memory message communication (lecture 6). Given a 6*6 processor array which are numbered as (i,j) (i,j=1,2,3,4,5,6) for a processor in row i and column j, arranged as a 2-dimensional torus, and assuming cut-through routing, *show the steps* in the following operations and obtain an estimate for the time taken s assuming the following parameters:

Startup time ts = 10 microseconds
Per hop time th =2 microseconds
Per byte time tw = 0.2 microseconds

a) **One-to-all broadcast** of 1000 bytes of data from processor (2, 2) to all the processors.

Ans.

Steps pre row from (2,2) Node :- 3 (i.e $\frac{\sqrt{p}}{2}$).

Total number of steps in row and column = $2(\frac{\sqrt{p}}{2}) = \sqrt{p}$

In One to all broadcast message size remain same = m = 1000.

Number of link traverse in row (l) = $\sqrt{p} - 1$

$$t_{comm} = 2\left(\sum_{i=1}^{\frac{\sqrt{p}}{2}}(ts + mtw) + lth\right)$$

$$t_{comm} = 2(\sqrt{p}/2\ t_s + \sqrt{p}/2\ m\ t_w + \sqrt{p} - 1(t_h))$$

$$t_{comm} = \sqrt{p}\ t_s + \sqrt{p}\ m\ t_w + 2\sqrt{p} - 1(t_h)$$

number of processor P = 36.
Message size m= 1000

Startup time ts = 10 microseconds
Per hop time th =2 microseconds
Per byte time tw = 0.2 microseconds

So,

$t_{comm} = 6\ t_s + 6\ m\ t_w + 2(5)\ t_h$
$t_{comm} = 6\ (10) + 6(1000)(0.2) + 10(2)$

t_{comm} = 1280 microsecond.

- In mash topology if we can start broadcast from any node it will not impact the communication time as number of steps in row and column remain same in one-to-all broadcast and message size and number of link traverse also remain same for every node.

Riten Chhatrala
rchhatrala@hawk.iit.edu
CWID:-A20385102

b) **All-to-all scatter** of sections of 1000 bytes from each processor to every other processor (i.e. A total of 6*6*1000 bytes of data exchanged overall)

Ans.

Steps pre row from:- 3 (i.e $\frac{\sqrt{p}}{2}$).

Total number of steps in row and column = 2($\frac{\sqrt{p}}{2}$) = $\sqrt{p}$

In all to all broadcast in every row average 14 messages sent.

Total message = 14m

Number of link traverse in row (l) = $\frac{\frac{\sqrt{p}}{2}}{\sqrt{p}}$) = 18

$$t_{comm} = 2\left(\sum_{i=1}^{\frac{\sqrt{p}}{2}}(ts + 14mtw) + lth\right)$$

$t_{comm} = 2(\sqrt{p}/2\ t_s + \sqrt{p}/2 * 14 * m\ t_w + 18 * (t_h))$

$t_{comm} = \sqrt{p}\ t_s + \sqrt{p}\ 14\ m\ t_w + 2 * 18 t_h)$

number of processor P = 36.

Message size m= 1000

Startup time ts = 10 microseconds
Per hop time th =2 microseconds
Per byte time tw = 0.2 microseconds

So,

$t_{comm} = 6\ t_s + 6 * 14\ m\ t_w + 36\ t_h$
$t_{comm} = 6\ (10) + 6*14*(1000)(0.2) + 36*(2)$

t_{comm} = 16932 microsecond.

Q-2. Modify get_data.c to make the largest rank process will collect data and final output.

Ans.

- I ran the steps as provided in the question sheet.
  1. First I login to Jarvis machine.
  2. Than compile my code and generate the executable file by running below command.
     a. **mpicc -c get_data.c**
     b. **mpicc -o get_data get_data.o**

rchhatrala@hawk.iit.edu
CWID:-A20385102

3. After that I make .bash file to run mpi program on multiple node in queue.
4. By running below command I scheduled the jobs in Jarvis cluster.
    a. **qsub –pe mpich 2 run_get_data.bash**
5. Run below command to monitor my job status.
    a. **qstat –u iit-username**

Finally as job get finished in will generate 2 output files and 2 error files. These 4 files are corresponds to applications standard error and output and parallel environment standard error and output files.

- I modify the get_data.c to **modify_get_data.c.** In **modify_get_data.c** I changed the sender process and receive process to largest rank process. So in **modify_get_data.c** program largest process will receive the data and final output.

  Modified file:- **modify_get_data.c**
  Modified .bash file:- **modify_get_data.bash**

  After running this modified program as following above steps generated output files are as below.
  Gerated output files:
  1. modify_get_data.bash.e17599
  2. modify_get_data.bash.o17599
  3. modify_get_data.bash.pe17599
  4. modify_get_data.bash.po17599

  I attach above file in zip folder.

Q-3.

Part – A Ans.

I have uploaded all my code file in Jarvis under home/rchhatrala/hw3 directory.

Dependency In Gaussian Elimination Serial Algorithm For loop is as below:

- The values in the outer most row and column are used to update the inner sub matrix before going on to next row/column, meaning that outer most loop has data dependency. Therefore, we can't use this loop because of data dependence to parallelize.
- Second loop does not have any dependency in any iteration so can parallelize this loop.
- Third inner most loop also does not have any dependency so can also parallelize this loop.

- **MPI Gaussian Elimination Implementation:-**
  File Name :- gaussMPI.c
    - In this program process 0 will first generate the random matrix of given size. Then in gaussEliminationMPI() function we are solving the linear equation with gauss elimination. Here every time I will broadcast the pivot row to every other processor and then process 0 will send part of row wise data to all other processor by static interleaved scheduling.

rchhatrala@hawk.iit.edu
CWID:-A20385102

- After this send every processor will get there part of work(rows) and they perform upper triangular computation on this received rows and send this updated rows back to the process 0.
- So for every outer most loop iteration(pivot row) this process will be followed.
- Here to utilize every processor I also assign part of row to process 0 as well instead of doing communication only.

- **Code Explanation**:-
  - **GenerateMatrix** :- This function will generate a NxN+1 matrix with random values. Here this matrix also include final column extra that is vector B in A*x =B.
  - **PrintMarix**:- By calling this function we will prinf the matrix. But we are currently not using this function as matrix is too large to print if we pass order of matrix large value.
  - **BackSubstitution**:- This function use to calculate answer of X vector after matrix will transform into upper triangular form.
  - **gaussEliminationMPI**:- This function will calculate the upper triangular matrix. It will preform computation as explain above.
  - **Main()**:- Execution start from this function. Dynamically allocate memory and generate NxN+1 matrix and Nx1 vector and then call above function and calculate time for running Gaussian Elimination and display it by process 0 only.

- **Running instruction:-**

1. **vi gaussMPI.c**
   - copy **gaussMPI.c** file from provide code and past it in terminal.
   - Save it by **:wq**
   - le is on Jarvis under rchhatrala/hw3 directory.

2. **mpicc -c gaussMPI.c**
   - compile the code by above command. This command will generate object file **gaussMPI.o**

3. **mpicc -o gaussMPI gaussMPI.o**
   - By running above command it will generate executable **gaussMPI.**

4. **mpirun -n 8 ./gaussMPI**
   - Run the program with 8 processor by running above command.

I am not able to run this code with multiple node by scheduling in queue with .bash file as my all jobs are going into qw state i.e queue waiting list. After that jobs are not getting out of this state. So I ran program with mpirun command only.

Output:

Program will output the elapsed time of execution. Here elapsed time is the time when processor 0 start sending data to all other processor and received the final result at processor 0.

Riten Chhatrala
rchhatrala@hawk.iit.edu
CWID:-A20385102

Other Implementation:

I also implemented this same code with MPI_Isend function.

File_Name:- gaussMPIIsend.c

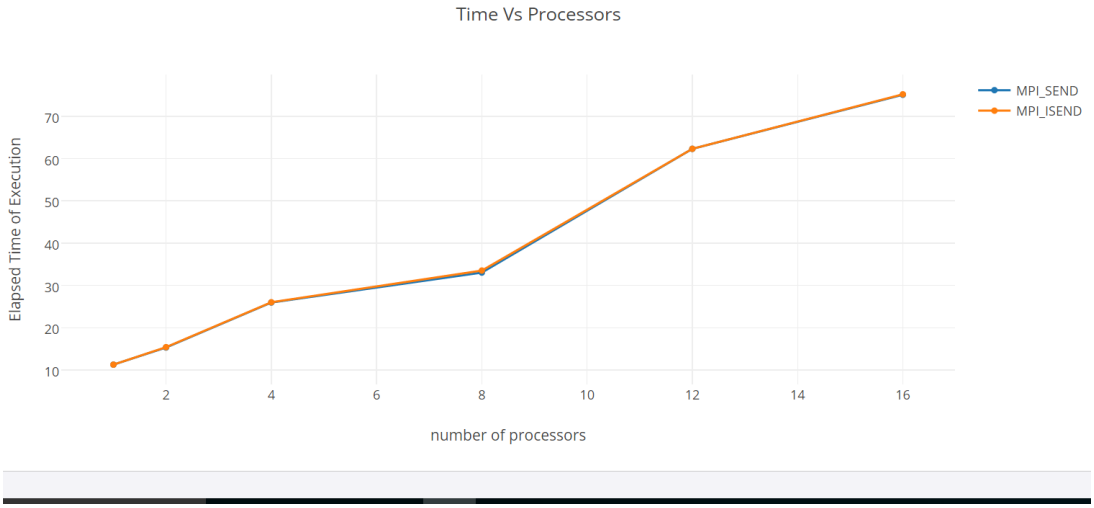
Part-B Ans.

- Running time of MPI algorithm with different number of processor.

Matrix Size :- 2000x2000

On Jarvis Machine:

Number of Processors	gaussMPI(with MPI Send)	gaussMPIIsend(MPI with Isend)
1	11.2751	11.2775
2	15.3246	15.4102
4	25.9590	26.0212
8	33.0406	33.5149
12	62.2871	62.2951
16	75.0748	75.1827



Conclusion :- So from graph and above table we can see that the time is increasing with the number of processor will increases. We can also see that the time for MPI_SEND and MPI_ISend is almost same.