SINGAPORE-MIT ALLIANCE
COMPUTATIONAL ENGINEERING PROGRAMME
CME5232: Cluster and Grid Computing Technologies for Scientific Computing
COMPUTATIONAL LAB No.3

17th July, 2009

PARALLEL PROGRAMMING USING MPI

Objectives

To experiment MPI library functions focusing on point to point and collective communications.

Background Knowledge

C/C++ programming languages and basis MPI functions.

Exercises (Due date: 21th July, 2009)

Please submit a short format report recording your observations based on the assigned exercises and programming assignments in a week's time. The report should be complete and clearly written to show what you have done and what you have observed and interpreted from these exercises. All equations, symbols etc. if any must be declared or explained clearly.

Question 1. Matrix - Vector multiplication

Consider a dense square matrix of order $n \times n$, a vector of order $n \times 1$ and number of processors p. Assume that n is divisible by p. This matrix is stored in row major order in a file. your are required to developed a parallel program to carry out a matrix-vector multiplication as follows;

a. Rowwise block-striped partitioning.

In this question, you are required to write a function that reads the dense square matrix and a column vector in. Processor 0^{th} should read in the number of n, then broadcast n to all other processors. Processor 0^{th} should then read in a block of n rows and distribute sub-blocks of n/p rows to each processor such that the first n/p rows go to processor 0^{th}, the next n/p go to processor 1^{th} and so on.

Once the distribution of the matrix is done, perform the parallel matrix-vector multiplication and print the results of the matrix-vector multiplication into an output file called *rowresult.txt.*

b. Columnwise block-striped partitioning.

In this question, you are required to write a function that reads the dense square matrix and column vector in. Processor 0^{th} should read in the number of n, then broadcast n to all other processors. Processor 0^{th} should then read in a block of n columns and distribute sub-blocks of n/p columns to each processor such that the first n/p columns go to processor 0^{th}, the next n/p go to processor 1^{th}

and so on. Similarly, processor 0^{th} distribute the n/p elements of column vector to each processor, such that the first n/p elements go to processor 0^{th}, the next go to processor 1^{th} and so on.
Once the distribution of the matrix and vector is done, perform the parallel matrix-vector multiplication and print the results of the matrix-vector multiplication into an output file called *columnresult.txt*.

c. Block-checkerboard partitioning.
Repeat the question (Q1.a) for the matrix vector multiplication using block-checkerboard partitioning method. State the addition requirements for n and p in order to use block-checkerboard partitioning. Here, you are require to use collective communication functions as much as you can.

d. Observation
Comment on the differences among the rowwise block-striped partitioning, columnwise block-striped partitioning and the block-checkerboard partitioning in the matrix-vector operation (Specify the main points including the advantages and disadvantages of each approach). Demonstrate all of the the above by explaining what you have done, by writing a program, executing the program on the SMA Hydra Cluster with the given sample file *matrix-vector-16.txt* of $n = 16$ using p=4, and showing the sample outputs. Measure the timing required to compute the matrix-vector multiplication for $n = 16, 32, 64, 128, 256, 512, 1024$ with $n = 16$. Plot the speed up and comment on the timings of each method.

Question 2. Matrix - Matrix multiplication
a. Given a serial code (*serial-matmul.c*) and a parallel implementation code of the Fox's algorithm (*fox.c*) for matrix-matrix multiplication involve square matrix A and B. you are now require to compile and run the serial code and parallel code with two square matrices A and B with order of (512×512), the parallel code using 4, 8 9, 16, 25 and 36 processors. Record and plot the required timing. Plot speed up of parallel code. Discuss on the observations.

b. When we discussed the Fox's algorithm, we observed that if it would be unrealistic to expect the system to provide n^2 physical processors. so we modified our original algorithm so that each processor stored sub-matrices of order $n/\sqrt{p}$.
Our basic algorithm in which we store a single row on each processor is also unrealistic. Outline a modification to the basic algorithm so that it stores a block of n/p rows on each processor. That means that processor 0^{th} store the first n/p of rows of A and B; processor 1^{th} stores the second n/p of rows of A and B, and so on.
At stage 0, for example, processor 0^{th} broadcast its n/p rows of B to the other processors. Each processor then multiplies the first n/p columns of its local matrix A to the local matrix from process 0^{th} to form a partial sum of the matrix-matrix product. This finishes stage 0. At stage 1, this procedure is repeated for the local part of matrix B of processor 1^{th} and the second n/p of columns of the local part of matrix A in each processor. We repeat this procedure until the stage $(p - 1)$.

In the end, processor 0^{th} print the result of the product of matrix A and B.

Compare the storage requirements of the modified Fox's algorithm and the modified basic algorithm. Implement the modification of the basic algorithm mentioned above. Carry out the parallel matrix-matrix multiplication of the two square matrices by using modified basic algorithm and *fox.c*. Compare the execution time of this modified basic algorithm to those of modified Fox's algorithm, plot the results out and give the discussions on the observations.