SINGAPORE – MIT ALLIANCE
COMPUTATIONAL ENGINEERING PROGRAM

CME5232: Cluster and Grid Computing Technologies and Tools for Scientific Computing
COMPUTATIONAL LAB No.3
17th July, 2009

PARALLE PROGRAMMING USING MPI

Objectives
To experiment MPI library functions focusing on point to point and collective communications, data decomposition.

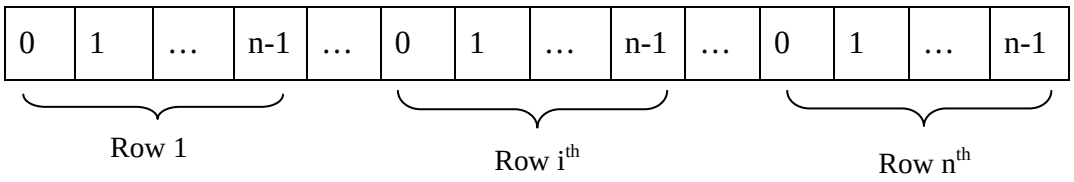
Background Knowledge
C/C++ programming languages and basis MPI functions.

Question 1. Matrix – Vector multiplication
Given a dense square matrix of order $n \times n$, a vector $n \times 1$ and p processors with the assumption n is divisible by p. The matrix is stored in row major order in a file. You are required to develop a parallel program to carry out a matrix – vector multiplication as follows:

1. Row-wise block striped partitioning
2. Column-wise block striped partitioning
3. Block-checker board partitioning

a) Analyse the question

- Dense square matrix is a matrix with dimension of $n \times n$, where, the almost elements are not equal to 'zero'.
- n is divisible by p, or another word, each processor will store n/p rows or n/p columns or sub-matrix with the order of $\left(n/\sqrt{p} \times n/\sqrt{p}\right)$ of the given matrix.
- Matrix is stored in row major order



b) Sequential algorithm

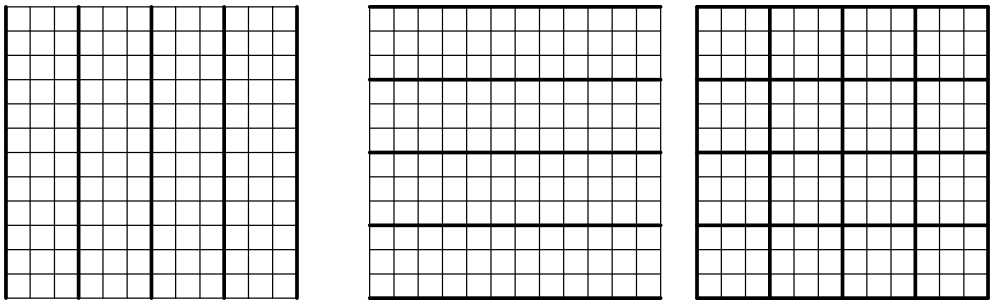
Input: a[0..n-1, 0..n-1] Matrix with dimension $n \times n$

```
                b[0..n-1]          Vector with dimension n × 1
Output:         c[0..n-1]          Vector with dimension n × 1
for (i = 0; i <= n-1; i++){
        c[i] = 0.0;
        for(j = 0; j <= n-1; j++){
                c[i] = c[i] + a[i][j] × b[j];
        }
}
```

c) Data decomposition matrix A

There are three straight ways to decompose an matrix A with order $n \times n$:

- **Row-wise block stripping**: each of processes is responsible for a contiguous group of (n/p) rows of the matrix. (see the following figure)
- **Column-wise block stripping**: Each of processes is responsible for contiguous of (n/p) columns of the matrix. (see the following figure)
- **Checked-board block stripping**: The process form a virtual grid, and the matrix is divided into 2D block aligning with grid. Assume the p processors form a grid with $\sqrt{p}$ rows and $\sqrt{p}$ columns. Each process is responsible for a block of (n/$\sqrt{p}$) rows and (n/$\sqrt{p}$) columns. (see the following figure)



Column-wise Row-wise Checkerboard block

d) Distribute vector b and c

There are two ways to distribute vector b and c. The Vector elements maybe replicated, meaning all the vector elements are copied on all the processors, or vector elements maybe divided among all the processes. In a block decomposition of an n-element vector, each of the p processes is responsible for a contiguous of (n/p) vector elements.

e) Method of communication

In this question, you are freely to choose whether **point to point communication** or **collective communication** to distribute and send data among processors. In order to improve the communication time, you are encouraged to group data into a single message, where the grouped data items must be stored in contiguous memory location.

To understand how data are stored in the memory loation by using C/C++ languages, the following code has been generated;

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
using std::cout;
using std::endl;
void understandDoubleArray(int N);
void understandequivalentSingleArray(int N);

int main(int argc, char **argv)
{
        Int N =4;
        understandDoubleArray(N);
        cout<<endl;
         void understandequivalentSingleArray(N);
        return 0;
}
void understandDoubleArray(int N);
{
        int ** A;
        int i,j;

        A = New int *[N];
        for(i=0; i<N;i++)
                A[i] = new int[N];
        For (i=0; i<N; i++)
                For(j=0; j<N; j++)
                        A[i][j] = i*N + j;
        Cout<<"Double Array"<<endl;
        Cout<<A<<endl;
        Cout<<&(A[0])<<endl;
        Cout<<A[0]<<endl;
        Cout<<&(A[0][0])<<endl;
        Cout<<A[0][0]<<endl;
        Cout<<&(A[0][N-1]);
        Cout<<A[0][N-1]<<endl;
```

```
void understandequivalentSingleArray(int N);
{
        int *A;
        int i,j;
        A = new int[N*N];
        for(i=0;i<N;i++)
                for(j=0; j<N; j++)
                        A[i*N+j] = i*N + j;
        Cout<<"Single Array"<<endl;
        Cout<<A<<endl;
        Cout<<&(A[0])<<endl;
        Cout<<A[0]<<endl;
        Cout<<&(A[N])<<endl;
        Cout<<A[N]<<endl;
```

f) You are provided a program called "***squareMatrixVectorGenerator.cpp"*** to generate matrices and vectors with difference sizes.

g) Procedure to do

g1) Row-wise block stripping

- Process "0" reads N, matrix A, vector b.
- Process "0" broadcasts N to all processor.
- Processor "0" sends N/p rows to corresponding processors.
- Processor "0" sends N vector elements of b to corresponding processors.
- Each processor computes their part for Matrix – Vector multiplication.
- All processors gather their results.

Note: Do time measurement for this method.

g2) Column-wise block stripping

- Process "0" reads N, matrix A, vector b.
- Process "0" broadcasts N to all processor.
- Processor "0" sends N/p columns to corresponding processors.
- Processor "0" sends N/p vector elements of b to corresponding processors.
- Each processor computes their part for Matrix – Vector multiplication.

- All processors gather their results.

Note: Do time measurement for this method.

g3) Checkerboard block stripping

- Process "0" reads N, matrix A, vector b.
- Process "0" broadcasts N to all processor.
- Processor "0" sends sub-matrix with order of $(n/\sqrt{p}) \times (n/\sqrt{p})$ to corresponding processors.
- Processor "0" sends N/p vector elements of b to corresponding processors.
- Each processor computes their part for Matrix – Vector multiplication.
- All processors gather their results.
- Processors compute the final result.

Note: Do time measurement for this method.

h) Show the result for three method with the size of matrix of N = 16.

i) Plot the pair of (timing . vs. matrix size) of the three methods to compare. Plot the (speedup .vs. the matrix size) of the three methods to compare as well. Discuss on the result.

Question 2.

a) Study the given code

In this question, you are provide a code called **"squareMatrixMatrixGenerator.cpp"** to generate two square matrices. You are also given the **"fox.c"** code and **"serial_mat_mult.c"** code. We just only need to run the serial and the parallel codes to compare the time.

- Generate two matrices with the order of (512 × 512).
  (**$gcc squareMatrixMatrixGenerator.cpp –o squareMatrixMatrixGenerator**
  $ ./ squareMatrixMatrixGenerator)
- Run the code with different number of processors for parallel code.
- Run the serial code.
- Plot the timing .vs. number of processor
- Calculate and plot the speed up

b) Modified the code

You are required to parallelize the code "serial_mat_mult.c" code as the following:

- Processor “0” read the matrices (A and B), and N
- Processor “0” broadcasts N to all processors.
- Processor “0” sends N/p rows of matrix B to corresponding processors.
- Each processor computes their part for Matrix – Matrix multiplication at corresponding stage.
- Process “0” print out the result.

b1) Record the timing then compare with “Fox.c” code.

b2) discuss on the observation results.