LABORATORY MANUAL

CE/CZ4011 and CPE/CSC423 : Parallel Computing

No. 1 : All Pairs Shortest Path Problem in MPI

SESSION 2015/2016
SEMESTER 1

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

ALL PAIRS SHORTEST PATH PROBLEM IN MPI

1. OBJECTIVE

The objective of this assignment is to gain experience in writing message passing programs, to understand how to design a parallel algorithm and execute it using the MPI environment, and to analyze the performance of the all pairs shortest path problem for different problem sizes and different numbers of processors. MPI stands for Message Passing Interface and is a widely used standard for writing message-passing programs.

2. LABORATORY

SWLab 1A/1B, previously MM2 Lab (N4-01a-02).

3. EQUIPMENT

Cluster (Network of Workstations connected by 10G switch) running Linux operating system, X-windows, C and MPI.

4. INTRODUCTION

Travel maps often contain tables showing the shortest distance between pairs of cities. The distance from city *A* to city *B* is given by the value of the element in the row corresponding to *A* and column corresponding to *B*. The route between two cities often passes through other cities in the table. The all pairs shortest path problem is concerned with generating such a table.

The input to the problem may be expressed as a weighted, directed graph, where the vertices are cities and the edges are distances between cities that are directly connected. The weighed, directed graph may be represented by an $N \times N$ *adjacency matrix*, where N is the number of vertices. The value of matrix element i, j is the weight (distance) from vertex i to vertex j if this edge exists. A solution to the all pairs shortest path problem is a table showing the shortest path between all pairs of vertices. The length of the path is given by the sum of the weights on the edges that form the path.

An algorithm to solve the all pairs shortest path problem is Floyd's algorithm. This transforms the input adjacency matrix into a matrix containing the length of the shortest path between all pairs of vertices. A sequential version of Floyd's algorithm may be described by the following pseudocode and has a time complexity of O(N^3):

```
/* number of vertices N and N × N adjacency matrix a; for
(k = 0; k < N; k++)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            a[i, j] = min(a[i, j], a[i, k] + a[k, j]);
```

When the algorithm terminates, the matrix *a* contains the length of the shortest path between all pairs of vertices. For example, Figure 1 shows an input adjacency matrix and the corresponding solution. In the input matrix, -1 indicates there is no direct path from vertex *i* to vertex *j*. In the solution, -1 indicates there is no path from vertex *i* to vertex *j* even through other vertices.

Input Data

$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 2 & 5 & -1 & -1 & -1 \\ 1 & -1 & 0 & 7 & 1 & -1 & 8 \\ 2 & -1 & -1 & 0 & 4 & -1 & -1 \\ 3 & -1 & -1 & -1 & 0 & 3 & -1 \\ 4 & -1 & -1 & 2 & -1 & 0 & 3 \\ 5 & -1 & 5 & -1 & 2 & 4 & 0 \end{array}$$

Solution

$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 2 & 5 & 3 & 6 & 9 \\ 1 & -1 & 0 & 6 & 1 & 4 & 7 \\ 2 & -1 & 15 & 0 & 4 & 7 & 10 \\ 3 & -1 & 11 & 5 & 0 & 3 & 6 \\ 4 & -1 & 8 & 2 & 5 & 0 & 3 \\ 5 & -1 & 5 & 6 & 2 & 4 & 0 \end{array}$$

Figure 1

A sequential version of Floyd's algorithm to solve the all pairs shortest path problem can be found on the NTULearn course site in the folder **`Content > Labs > Lab 1 – MPI > APSP sample files >`** This version can be used to measure the sequential execution time and also as a reference to check the correctness of the result of the parallel algorithm. The folder also contains some utility functions to generate a random input adjacency matrix of any size and to check the result of the parallel algorithm by comparing against the sequential solution. An example test program is also given in this folder.

5. EXPERIMENT

5.1 Design and write the parallel program

Create a directory called **`LAB1`** and copy the test program into the **`LAB1`** directory.

Design a parallel algorithm for the All Pairs Shortest Path Problem using Foster's design methodology. Discuss the approach you adopt, taking into consideration the overhead of different partitioning and communication methods and different ways of agglomeration and mapping.

Based on your design, modify the test program by changing the second call of the sequential algorithm to a call to a routine that implements a parallel version of Floyd's algorithm using MPI. You may use either collective or point-to-point communication, but your report should explain the reasons for your choice. You need only implement one approach.

5.2 Benchmark the algorithms

Instrument both the sequential algorithm and the parallel algorithm to time their execution. The sequential and parallel algorithms may be timed by calling `gettimeofday` immediately before and after the algorithm, as follows:

```
struct timeval tv1,tv2;
...
gettimeofday(&tv1,NULL); //
code to be timed
gettimeofday(&tv2,NULL);
printf("Elasped time = %ld usecs\n", (tv2.tv_sec -tv1.tv

    _sec)*1000000+tv2.tv_usec-tv1.tv_usec);
```

Note that `gettimeofday` gives the current time in seconds and microseconds since 00:00:00 UTC, January 1 1970. For example:

```
Start time    = 1156935422, 782651
End time      = 1156935423, 34417
Difference    = 251766 usecs
```

For the parallel algorithm, start the timer immediately before distributing the data and end the timer immediately after collecting the results. Compare the two algorithms by measuring the execution time for different numbers of vertices *N*. For each value of *N*, measure the execution time for the same adjacency matrix on different numbers of processors *p* (e.g. *p* = 2, 4, 6, 8, 10). For simplicity, you can use values of *N* which are exactly divisible by *p* (e.g. *N* = 1200, 2400, 4800). Show the results as graphs and provide a full analysis.

6. REPORT

Results of this assignment should be recorded in your team report. The report should include the source code listings, results of the programs, the timings shown as graphs, and a full description and discussion of the work.

7. APPENDIX

Functions are provided for generating random input adjacency matrices, comparing the values of two arrays and computing the All Pairs Shortest Path using a sequential algorithm. The functions are declared in `MatUtil.h` and defined in `MatUtil.c`. To use these functions, simply include `MatUtil.h` in your source code. Details of the functions are provided as comments in `MatUtil.h`.

Suppose you have allocated three integer arrays, `int *mat; int *ref; int *result` with sizes of `sizeof(int)*N*N`.

- To generate a random input matrix:

```
GenMatrix(mat, N);
```

- To execute the sequential algorithm to obtain the correct answer:

```
memcpy(ref,   mat,   sizeof(int)*N*N);
ST_APSP(ref, N);
```

- Suppose the result of the parallel algorithm with input `mat` is in `result`. To compare the result with the sequential solution (`isCorrect` should be `true`):

```
bool isCorrect=CmpArray(ref,result, N*N);
```

- To compile and run the sequential `APSPtest.c` program (*N* is number of vertices):

```
$ gcc -O3 -std=c99 -o APSPtest APSPtest.c MatUtil.c

$ ./APSPtest N
```

- To compile and run the `APSPtest.c` program after converting to MPI code:

```
$ mpicc -O3 -std=c99 -o APSPtest APSPtest.c MatUtil.c

$ mpirun -np numberofprocessors -machinefile machines -mca btl
  tcp,self,sm APSPtest N
```

8. REFERENCES

[1] Lecture Notes, particularly lectures on Distributed Memory Programming.
[2] Barry Wilkinson and Michael Allen. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice Hall (2005), particularly Chapter 2.
[3] NTULearn course site.