LABORATORY MANUAL

CE/CZ4011 and CPE/CSC423 : Parallel Computing

No. 2 : All Pairs Shortest Path Problem in OpenMP

SESSION 2016/2017
SEMESTER 1

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

ALL PAIRS SHORTEST PATH PROBLEM IN OPENMP

1. OBJECTIVE

The objective of this assignment is to gain an understanding of shared memory programming techniques and to become familiar with the OpenMP environment. OpenMP is an application programming interface for parallel programming on shared memory architectures. It consists of a set of compiler directives to a C compiler and a library of support functions.

2. LABORATORY

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

3. EQUIPMENT

Multicore workstation running Linux operating system, C and OpenMP.

4. INTRODUCTION

This assignment concerns the same problem as that in Assignment 1: how to generate a table showing the shortest path between all pairs of cities. 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 there exists an edge between these two vertices. 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 the Floyd-Warshall 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.

As in Lab 1, a sequential program for Floyd-Warshall and a program to generate a random adjacency graph are available. They can be found in on the NTULearn course site in the folder **`Content > Labs > Lab 2 – OpenMP > APSP sample files`**. You should compare the correctness and running time of your parallel program with that of the sequential program.

5. EXPERIMENT

5.1 Design and write the parallel program

Design a parallel algorithm for the All Pairs Shortest Path Problem, considering alternative ways of exploiting parallelism and how these could be expressed in OpenMP. Based on your design, modify the test program to implement a parallel version of Floyd's algorithm using OpenMP directives. In implementing your program, you should pay particular attention to the use of private and shared variables and avoid race conditions. Comment on any techniques used to improve the efficiency of your code. You need only implement one approach.

5.2 Benchmark the algorithms

Instrument both the sequential algorithm and the parallel algorithm to time their execution. 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 with different numbers of threads *p* (e.g. *p* = 2, 4, 6, 8, 10). 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 and speedup shown as graphs, and a full description and analysis of the work, including a discussion of the performance results.

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 (single thread) 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 your OpenMP program, do the following:

  ```
  $ gcc –o APSPtest –fopenmp APSPtest.c MatUtil.c
  ```

  ```
  $ ./APSPtest N
  ```

- By default 12 threads will be used (6 processing cores with hyperthreading). To set the number of threads to *p* before execution, type:

  ```
  $ export OMP_NUM_THREADS=p
  ```