**LAB 2 REPORT**

**OpenMP (“Floyd’s Algorithm”)**

**Précis:**

G: Graph = {v1, v2, e, w}

T: threads = {1,2,4,8,12,16}

N: # of nodes = 2000

S: seed = 10;

Upon running the program, key differences in time were noticed upon thread increase. Multiple tests were run with **T** amount of threads and N amount of nodes, N being a constant of 2000 nodes. The seed was a random # ranging from 1-10 to fill in **G**. The Graph isrepresented asdistance [i][j], the path from node to node. Floyd’s algorithm solves the shortest path problem and upon finding a shorter path replaces the value within the 2D matrix with the shorter path value.

Using the **omp\_get\_wtime(),** the time it took to complete the task was based on the number of threads exported. The greater number of threads increased the overall effectiveness of the code. Each test displayed a major difference in time in seconds as shown in **Figure1**.

**Processes:**

icc –o test –fopenmp test.c

export OMP\_NUM\_THREADS= 1

./test

**C – Programming Implementation of Floyd’s Algorithm**

//FW\_AL: Finds the shortest path, if not will put 0.

void FW\_Al() {

int i, j, k;

for (k = 0; k < n; ++k)

#pragma omp parallel for private(i,j) //Parallelization

for (i = 0; i < n; ++i)

for (j = 0; j < n; ++j)

/\* 1. If i and j are different nodes and

2. If the paths between i and k and between

k and j exist, then do\*/

if ((distance[i][k] \* distance[k][j] != 0) && (i != j))

/\* Check to get a shorter path

between i and j by placing

k somewhere on the current

path \*/

if ((distance[i][k] + distance[k][j] < distance[i][j]) || (distance[i][j] == 0))

distance[i][j] = distance[i][k] + distance[k][j];

**Results:**

**Test 4**

Nodes: 2000

Seed: 10

Threads: 8

Time: 2.223078 seconds

**Test 5**

Nodes: 2000

Seed: 10

Threads: 12

Time: Time: 2.225224 seconds

**Test 6**

Nodes: 2000

Seed: 10

Threads: 16

Time: 2.470988 seconds

**Test 1**

Nodes: 2000

Seed: 10

Threads: 1

Time: 13.428659 seconds

**Test 2**

Nodes: 2000

Seed: 10

Threads: 2

Time: 7.619810 seconds

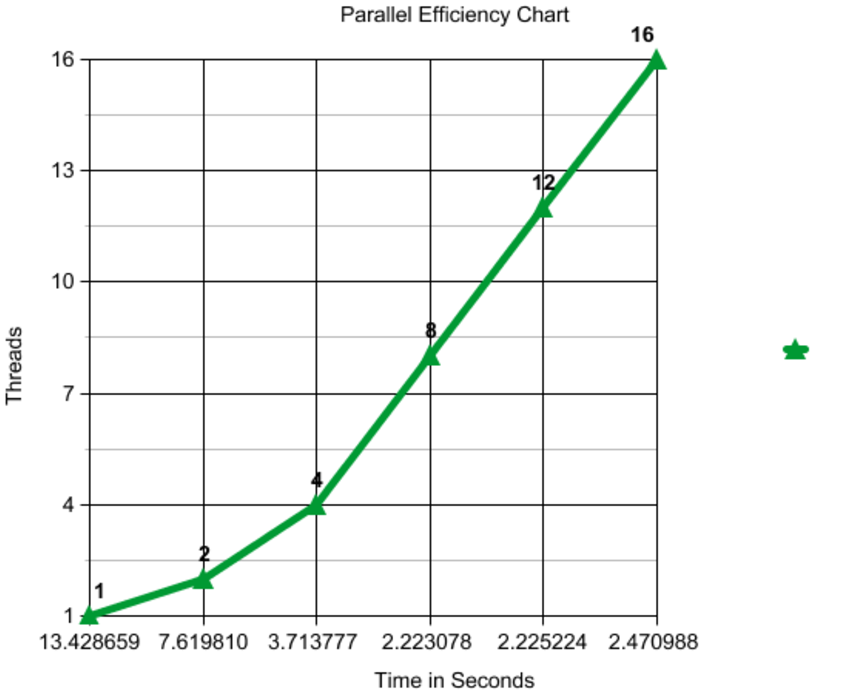
**Test 3**

Nodes: 2000

Seed: 10

Threads: 4

Time: 3.713777 seconds

**Figure1.**

**Code**

/\*

\* Created on: 4 March 24th 2017

\* Author: Thomas Diaz, Fotis Tsakiris

\*/

#include <stdio.h>

#include <omp.h>

#include <stdlib.h>

#define n 2000

//#define n 500

//#define n 1000 /\* # of nodes \*/

//distance[i][j] array is the distance between i & j if it exist, otherwise 0.

int distance[n][n];

//int iterate[n\*n];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printDistance(int x) {

int i, j;

printf(" ");

for (i = 0; i < x; ++i)

printf("%4c", 'A' + i);

printf("\n");

for (i = 0; i < x; ++i) {

printf("%4c", 'A' + i);

for (j = 0; j < x; ++j)

printf("%4d", distance[i][j]);

printf("\n");

}

printf("\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printE\_V\_W(int x) {

int i, j;

printf(" PATH Distance \n");

int count = 1;

for(i=0;i<x;++i){

for(j=0;j<x;++j)

// printf("%4c ->%4c %4d\n",'A'+i,'A'+j,distance[i][j]);

printf("%4d to%4d %4d\n",count+i,count+j,distance[i][j]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//FW\_AL: Finds the shortest path, if not will put 0.

void FW\_Al(int x) {

int i, j, k;

//FW\_AL(int n){

for (k = 0; k < x; ++k)

#pragma omp parallel for private(i,j)

for (i = 0; i < x; ++i)

for (j = 0; j < x; ++j)

/\* 1. If i and j are different nodes and

2. If the paths between i and k and between

k and j exist, then do\*/

if ((distance[i][k] \* distance[k][j] != 0) && (i != j))

/\* Check to get a shorter path

between i and j by placing

k somewhere on the current path \*/

if ((distance[i][k] + distance[k][j] < distance[i][j]) || (distance[i][j] == 0))

distance[i][j] = distance[i][k] + distance[k][j];

// weight[i] += distance[i][j];

// iterate[n] = k;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[]) {

/\*This method creates a matrix and fills in distance[][]\*/

double start,stop; // time it takes to generate using Floyd’s Algorithm.

int i,j,x;

printf("Please Enter the # of Nodes you'd like to run\n");

scanf("%d,",&x);

for (i = 0; i < x; ++i)

for (j = 0; j < x; ++j)

if (i==j)

distance[i][j]=0;

else

distance[i][j]= (int)( 11.0 \* rand() / ( RAND\_MAX + 1.0 ));

printf("\n");

printf("Matrix A\n ");

// printDistance();// prints the matrix

printE\_V\_W(x);

printf("\n");

printf("Matrix B | FLOYD's Algorithm\n ");

start = omp\_get\_wtime(); // get time

FW\_Al(x); //initializes Floyd’s Algorithm

stop = omp\_get\_wtime(); // stop time

// printDistance();// print the new matrix with shortest paths

printE\_V\_W(x);

printf("-----------------------------\n");

printf("Nodes: %d | Time: %f seconds\n",x,stop-start);

printf("-----------------------------\n");

return 0;

}