Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы № 6

по дисциплине Теория параллельных алгоритмов

Тема: Алгоритм Флойда поиска кратчайших путей между всеми вершинами графа

Выполнила: ст. гр. 36/1

Фролова А.А.

Проверила: Патыковская М.В.

Краснодар

2024

**Цель работы:**

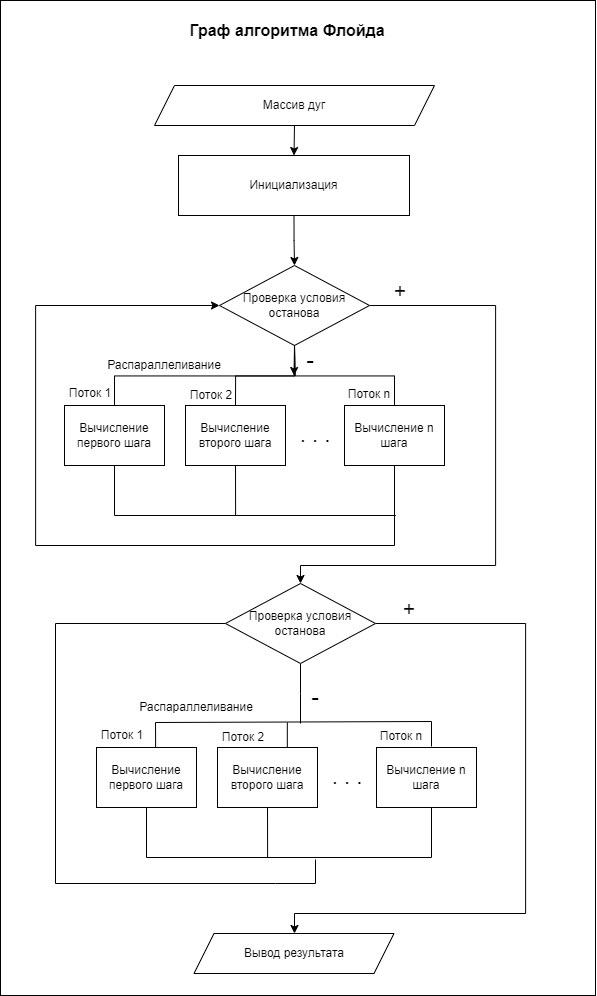
Построить последовательную и параллельную программы для поиска кратчайших путей между всеми вершинами взвешенного графа с помощью алгоритма Флойда. Проследить зависимость времени работы от количества потоков и ядер.

**Ход работы.**

Данные, полученные в результате экспериментов в сек:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 500 | 0.27 | 0.15 | 0.09 | 0.07 | 0.06 | 0.07 | 0.06 | 0.06 | 0.07 | 0.06 |
| 1000 | 2.21 | 1.11 | 0.74 | 0.58 | 0.45 | 0.37 | 0.39 | 0.50 | 0.45 | 0.42 |
| 2000 | 18.81 | 8.99 | 6.02 | 4.65 | 3.71 | 3.43 | 3.32 | 3.11 | 3.48 | 3.26 |

**Граф алгоритма**



**Листинг программы**

// Import header files for program

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

//Define the number of nodes in the graph

#define N 10000

//Define minimum function that will be used later on to calcualte minimum values between two numbers

#ifndef min

#define min(a,b) (((a) < (b)) ? (a) : (b))

#endif

//Define matrix of size N \* N to store distances between nodes

//Initialize all distances to zero

int distance\_matrix[N][N] = {0};

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

{

int nthreads;

int src, dst, middle;

//Initialize the graph with random distances

for (src = 0; src < N; src++)

{

for (dst = 0; dst < N; dst++)

{

// Distance from node to same node is 0. So, skipping these elements

if(src != dst) {

//Distances are generated to be between 0 and 19

distance\_matrix[src][dst] = rand() % 20;

}

}

}

//Define time variable to record start time for execution of program

double start\_time = omp\_get\_wtime();

for (middle = 0; middle < N; middle++)

{

int \* dm=distance\_matrix[middle];

for (src = 0; src < N; src++)

{

int \* ds=distance\_matrix[src];

for (dst = 0; dst < N; dst++)

{

ds[dst]=min(ds[dst],ds[middle]+dm[dst]);

}

}

}

double time = omp\_get\_wtime() - start\_time;

printf("Total time for sequential (in sec):%.2f\n", time);

for(nthreads=1; nthreads <= 10; nthreads++) {

//Define different number of threads

omp\_set\_num\_threads(nthreads);

// Define iterator to iterate over distance matrix

//Define time variable to record start time for execution of program

double start\_time = omp\_get\_wtime();

/\* Taking a node as mediator

check if indirect distance between source and distance via mediator

is less than direct distance between them \*/

#pragma omp parallel shared(distance\_matrix)

for (middle = 0; middle < N; middle++)

{

int \* dm=distance\_matrix[middle];

#pragma omp parallel for private(src, dst) schedule(dynamic)

for (src = 0; src < N; src++)

{

int \* ds=distance\_matrix[src];

for (dst = 0; dst < N; dst++)

{

ds[dst]=min(ds[dst],ds[middle]+dm[dst]);

}

}

}

double time = omp\_get\_wtime() - start\_time;

printf("Total time for thread %d (in sec):%.2f\n", nthreads, time);

}

return 0;

}

Вывод:

В результате выполнения лабораторной работы №6 были созданы программы для поиска кратчайших путей между всеми вершинами взвешенного графа с помощью алгоритма Флойда. В ходе работы было установлено, что оптимальное количество потоков было примерно равно 8 ядер. Также можно отметить, что обе реализации довольно быстро работают при малой размерности систему, но последовательный значительно замедляется с ростом этого показателя. В дальнейшем полученные знания будут использованы для написания более сложных программ.