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 № 5

по дисциплине «Теория параллельных алгоритмов»

Выполнил:

Корнилов К.А.

Проверил: преподаватель

Нигодин Е. А.

Краснодар

2024

**Тема работы:** Алгоритм Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

**Цель:** Распараллеливание процесса нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе с помощью алгоритма Флойда – Уоршелла с использованием технологии OpenMP.

**Задание:**

1. Построить последовательный алгоритм. Построить граф алгоритма. Исследовать зависимости и возможности распараллеливания.
2. Написать параллельную реализацию алгоритма.
3. Составить таблицу, отражающая сравнительное время выполнения последовательного и параллельных алгоритма для разных массивов на 2-х и 4-х ядрах.

**Ход работы:**

1. Был составлен и написан на языке С++ последовательный алгоритм Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

Входными данными алгоритма выступают: матрица смежности графа размеров NxN.

Результатом выступает матрица размера NxN, в ячейках которой записаны длины кратчайших путей из вершины, соответствующей строке матрицы, в вершину, соответствующую столбцу матрицы.

Данная длины кратчайших путей вычисляются следующим образом: для каждой вершины графа запускается проверка путей в матрице. Если путь через данную вершину оказывается короче, чем уже записанный в ячейку путь, то в значение записывается новое значение пути.

Тогда последовательный алгоритм Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе на рисунке 1 для частного случая (4 вершины графа):

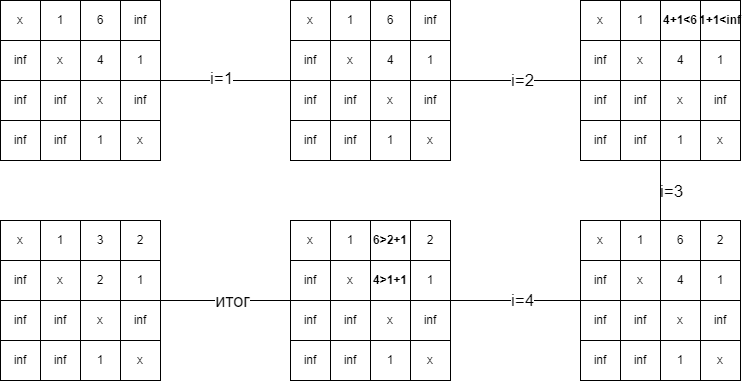


Рисунок 1 – Алгоритм Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе

Код реализованного алгоритма имеет следующий вид:

double\*\* seq\_fload\_yor(double\*\* matr, int N) {

double\*\* res = new double\* [N];

for (int i = 0; i < N; i++) {

res[i] = new double[N];

for (int j = 0; j < N; j++) {

if (matr[i][j] == 0 && i!=j) {

res[i][j]= numeric\_limits<double>::infinity();

}

else {

res[i][j] = matr[i][j];

}

}

}

for (int k = 0; k < N; k++) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (res[i][j] > res[i][k] + res[k][j]) {

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

}

}

}

}

return res;

}

Согласно схеме алгоритма сравнивания длину путей для отдельной вершины происходят независимо друг от друга. Тогда данные сравнения можно на каждой итерации работы выполнять параллельно. Схема параллельного алгоритма Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе представлена на рисунке 2.

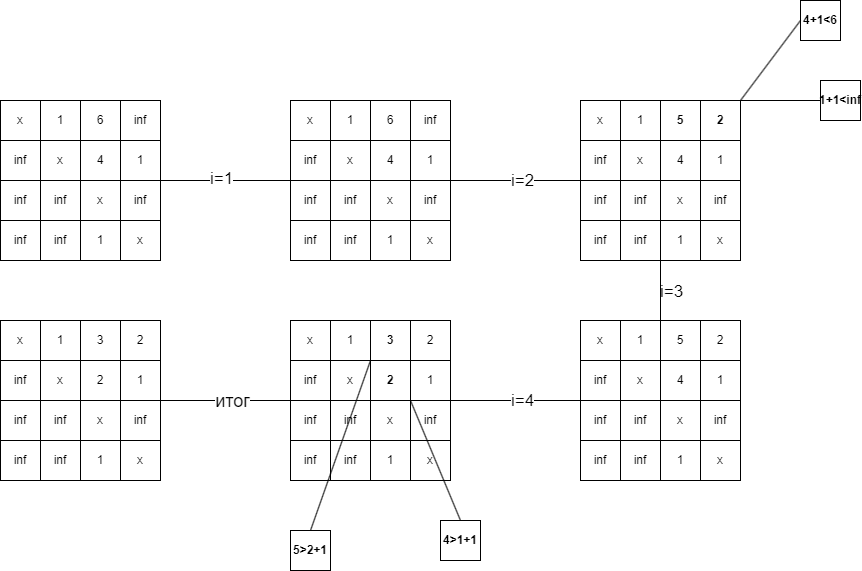


Рисунок 2 – Схема параллельного алгоритма Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе

1. Была написана параллельная реализация алгоритма Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе на языке С++ с использованием технологии OpenMP.

Код реализованного алгоритма имеет следующий вид:

double\*\* parallel\_fload\_yor(double\*\* matr, int N,int numberthreads) {

double\*\* res = new double\* [N];

for (int i = 0; i < N; i++) {

res[i] = new double[N];

for (int j = 0; j < N; j++) {

if (matr[i][j] == 0 && i != j) {

res[i][j] = numeric\_limits<double>::infinity();

}

else {

res[i][j] = matr[i][j];

}

}

}

for (int k = 0; k < N; k++) {

#pragma omp parallel for collapse(2) num\_threads(numberthreads)

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (res[i][j] > res[i][k] + res[k][j]) {

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

}

}

}

}

return res;

}

1. Используя написанные программы, была проведена оценка времени работы алгоритмов для различного размера матриц смежности и для различного количество ядер. Результаты выполненных оценок представлены в виде графика времени выполнения на рисунке 3.

Синим цветом изображено время работы последовательного алгоритма.

Оранжевым цветом изображено время работы параллельного алгоритма на 2 ядрах.

Серым цветом изображено время работы параллельного алгоритма на 3 ядрах.

Желтым цветом изображено время работы параллельного алгоритма на 4 ядрах.

Рисунок 3 – График зависимости времени работы программы от размера массива и количества ядер

**Вывод:** Было осуществлено распараллеливание процесса нахождения длин кратчайших путей алгоритма Флойда – Уоршелла с помощью технологии OpenMP и была осуществлена оценка времени работы данной программы. Согласно проведенными исследованиям максимальная скорость работы программы была достигнута на 4 потоках.

**ПРИЛОЖЕНИЕ.**

**Приложение 1. Полный код программы**

#include<iostream>

#include<string>

#include<sstream>

#include<omp.h>

using namespace std;

double\*\* input\_matr(int N, int M) {

double\*\* matr = new double\* [N];

for (int i = 0; i < N; i++) {

matr[i] = new double[M];

}

for (int i = 0; i < N + 1; i++) {

string inp = "";

getline(std::cin, inp);

stringstream ss(inp);

int j = 0;

string word;

while (ss >> word) {

matr[i - 1][j] = stod(word);

j = j + 1;

}

}

return matr;

}

void print\_matr(double\*\* matr, int N, int M) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cout << matr[i][j] << ' ';

}

cout << endl;

}

}

double\*\* rand\_matr(int N) {

double\*\* res = new double\* [N];

for (int i = 0; i < N; i++) {

res[i] = new double[N];

for (int j = 0; j < N; j++) {

res[i][j] = rand() % 30 + 1;

}

}

return res;

}

double\*\* seq\_fload\_yor(double\*\* matr, int N) {

double\*\* res = new double\* [N];

for (int i = 0; i < N; i++) {

res[i] = new double[N];

for (int j = 0; j < N; j++) {

if (matr[i][j] == 0 && i!=j) {

res[i][j]= numeric\_limits<double>::infinity();

}

else {

res[i][j] = matr[i][j];

}

}

}

for (int k = 0; k < N; k++) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (res[i][j] > res[i][k] + res[k][j]) {

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

}

}

}

}

return res;

}

double\*\* parallel\_fload\_yor(double\*\* matr, int N,int numberthreads) {

double\*\* res = new double\* [N];

for (int i = 0; i < N; i++) {

res[i] = new double[N];

for (int j = 0; j < N; j++) {

if (matr[i][j] == 0 && i != j) {

res[i][j] = numeric\_limits<double>::infinity();

}

else {

res[i][j] = matr[i][j];

}

}

}

for (int k = 0; k < N; k++) {

#pragma omp parallel for collapse(2) num\_threads(numberthreads)

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (res[i][j] > res[i][k] + res[k][j]) {

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

}

}

}

}

return res;

}

int main()

{

setlocale(LC\_ALL, "russian");

//double a, b;

//int N;

//cout << "Количество вершин графа массива" << endl;

//cin >> N;

//cout << "Введите матрицу смежности" << endl;

//double\*\* matr\_sm = input\_matr(N, N);

for (int N = 100; N <= 2000; N = N + 50) {

double\*\* matr\_sm = rand\_matr(N);

//print\_matr(matr\_sm, N, N);

cout << N << endl;

double start = clock();

double end = clock();

start = clock();

double\*\* res = seq\_fload\_yor(matr\_sm, N);

end = clock();

cout << "Матрица кратчайших путей" << endl;

//print\_matr(res, N, N);

cout << end - start << endl;

cout << "Кратчашиие для пар 2" << endl;

start = clock();

res = parallel\_fload\_yor(matr\_sm, N, 2);

end = clock();

//print\_matr(res, N, N);

cout << end - start << endl;

cout << "Кратчашиие для пар 3" << endl;

start = clock();

res = parallel\_fload\_yor(matr\_sm, N, 3);

end = clock();

//print\_matr(res, N, N);

cout << end - start << endl;

cout << "Кратчашиие для пар 4" << endl;

start = clock();

res = parallel\_fload\_yor(matr\_sm, N, 4);

end = clock();

//print\_matr(res, N, N);

cout << end - start << endl;

cout << endl;

}

}