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):

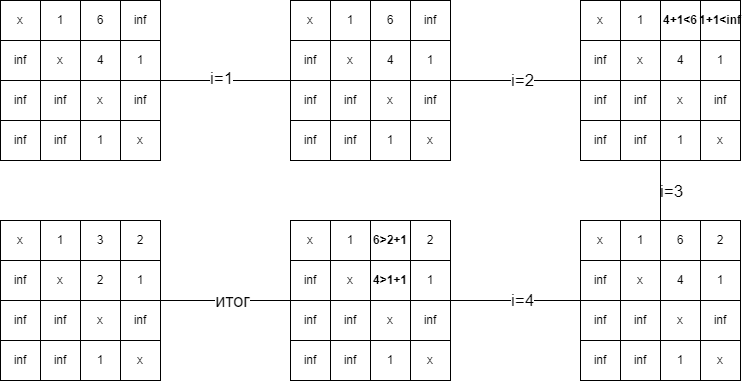


Рисунок 1 – Алгоритм Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе

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.

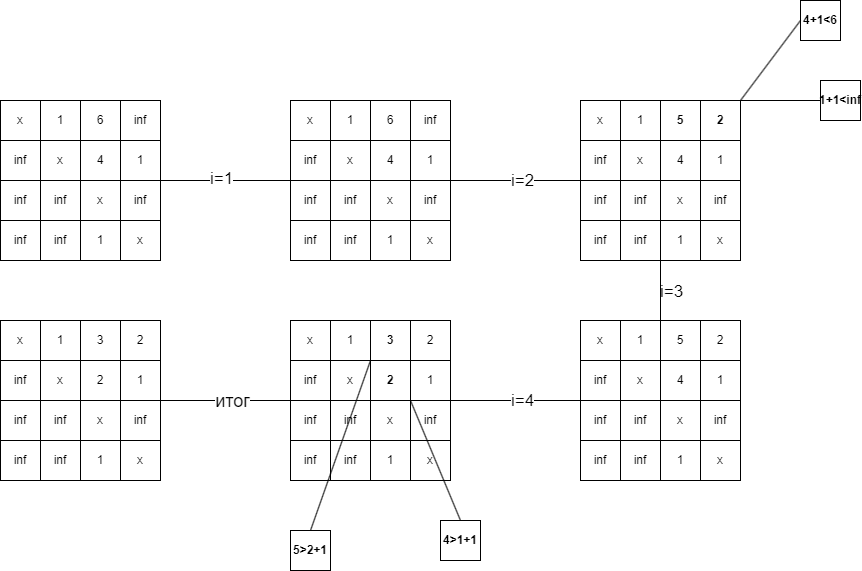


Рисунок 2 – Схема параллельного алгоритма Флойда – Уоршелла для нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе

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 массивов и для различного количество ядер. Результаты выполненных оценок представлены в виде графика времени выполнения на рисунке 3.

По оси Ox изображены количества элементов массива. По оси Oy изображено время работы алгоритма.

Голубым цветом изображено время выполнения для последовательного алгоритма пузырька.

Прозрачно – голубым изображено время выполнения для параллельного алгоритма пузырька на 2 ядрах.

Синим изображено время выполнения для параллельного алгоритма пузырька на 3 ядрах.

Темно – синим изображено время выполнения для параллельного алгоритма пузырька на 4 ядрах.

Зеленым цветом изображено время выполнения для последовательного алгоритма Шела.

Прозрачно – зеленым изображено время выполнения для параллельного алгоритма Шела на 2 ядрах.

Светло – зеленым изображено время выполнения для параллельного алгоритма Шела на 3 ядрах.

Темно – зеленым изображено время выполнения для параллельного алгоритма Шела на 4 ядрах.

Оранжевым цветом изображено время выполнения для последовательного алгоритма быстрой сортировки.

Прозрачно – оранжевым изображено время выполнения для параллельного алгоритма быстрой сортировки на 2 ядрах.

Светло – желтым изображено время выполнения для параллельного алгоритма быстрой сортировки на 3 ядрах.

Темно – желтым изображено время выполнения для параллельного алгоритма быстрой сортировки на 4 ядрах.

Рисунок 7 – График зависимости времени работы программы от размера массива и количества ядер

**Вывод:** Было осуществлено распараллеливание процесса сортировки массива алгоритмами пузырька, Шела, быстрой сортировки с помощью технологии OpenMP и была осуществлена оценка времени работы данной программы. Согласно проведенными исследованиям максимальная скорость работы программы была достигнута на 2 потоках.

**ПРИЛОЖЕНИЕ.**

**Приложение 1. Полный код программы**

#include<iostream>

#include<string>

#include<sstream>

#include<omp.h>

using namespace std;

double func\_to\_calc(double x) {

return x\*x;

}

double Simpson(double a, double b, double N) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double res = 0;

for (int i = 0; i < N-1; i=i+2) {

res += step / 3 \* (func\_to\_calc(X\_int[i]) + 4 \* func\_to\_calc((double)(X\_int[i + 1])) + func\_to\_calc(X\_int[i + 2]));

}

return res;

}

double Simpson\_parallel(double a, double b, int N,int num\_thr) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double res = 0;

#pragma omp parallel num\_threads(num\_thr) reduction(+:res)

{

#pragma omp for collapse(1)

for (int i = 0; i < N - 1; i = i + 2)

{

res += step / 3 \* (func\_to\_calc(X\_int[i]) + 4 \* func\_to\_calc((double)(X\_int[i + 1])) + func\_to\_calc(X\_int[i + 2]));

}

};

return res;

}

double Simpson\_parallel\_2(double a, double b, int N,int num\_thr) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double\* res\_f = new double[3];

double res = 0;

#pragma omp parallel num\_threads(num\_thr) reduction(+:res)

{

#pragma omp for collapse(2)

for (int i = 0; i < N - 1; i = i + 2)

{

for (int j = 0; j < 3; j++) {

res\_f[j] = func\_to\_calc(X\_int[i + j]);

}

res += step / 3 \* (res\_f[0] + 4 \* res\_f[1] + res\_f[2]);

}

};

return res;

}

int main()

{

setlocale(LC\_ALL, "russian");

double a,b;

int N;

cout << "Начало отрезка" << endl;

cin >>a;

cout << "Конец отрезка" << endl;

cin >> b;

cout << "Количество отрезков для разбиения" << endl;

cin >> N;

double start\_time = clock();

cout << Simpson\_parallel\_2(a, b, N, 2) << endl;

double end\_time = clock();

cout <<"Второй вид 2 ядра:"<<end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel(a, b, N, 2) << endl;

end\_time = clock();

cout << "Первый вид 2 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel\_2(a, b, N, 3) << endl;

end\_time = clock();

cout << "Второй вид 3 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel(a, b, N, 3) << endl;

end\_time = clock();

cout << "Первый вид 3 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel\_2(a, b, N,4)<<endl;

end\_time = clock();

cout << "Второй вид 4 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel(a, b, N,4)<<endl;

end\_time = clock();

cout << "Первый вид 4 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson(a, b, N) << endl;

end\_time = clock();

cout << "Последовательный вид:" << end\_time - start\_time << endl;

}