МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ  
  
Федеральное государственное автономное образовательное учреждение   
высшего образования   
«Самарский национальный исследовательский университет   
имени академика С,П, Королёва» (Самарский университет)

Институт информатики и кибернетики  
Кафедра программных систем  
  
  
Отчёт по лабораторному практикуму

по курсу «Параллельное программирование»

Тема «Параллельные вычисления в разделяемой памяти»

Вариант № 3,

Студенты: Мананников М.А.,

Горбачев И.Г.,

Рыбаков Р.С.

Группа: 6403-020302D  
Преподаватель: Востокин С.В.  
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2024

Цель работы: знакомство с основными приёмами параллельного программирования и исследования производительности параллельных программ,

Постановка задачи: требуется реализовать параллельный алгоритм поэлементного вычисления произведения квадратных матриц в разделяемой памяти. Сравнить производительность OpenMP - реализации и реализации, использующей API операционных систем.

Порядок выполнения работы:

1. На основе последовательного алгоритма матричного умножения разработать параллельный алгоритм решения той же задачи,
2. Реализация параллельной программы на языке C++ с использованием технологии OpenMP,
3. Реализация параллельной программы с помощью WinAPI,
4. Добавление в программы кода для анализа производительности, Измерить время работы программы для различных размеров матриц, Примерный набор значений размеров матриц (n): 600, 840, 984, 1128, 1488, 1992, 3000. Построение зависимости ускорения и эффективности параллельной программы от размера решаемой задачи,
5. Оформление отчёта, Отчет должен содержать графики, построенные по данным из таблиц,

РЕЗУЛЬТАТЫ

Ниже предоставлены результаты работы программы. На рисунках 1-2 представлены зависимости времени и коэффициента ускорения от размерности матрицы на процессоре M1-ARM64.

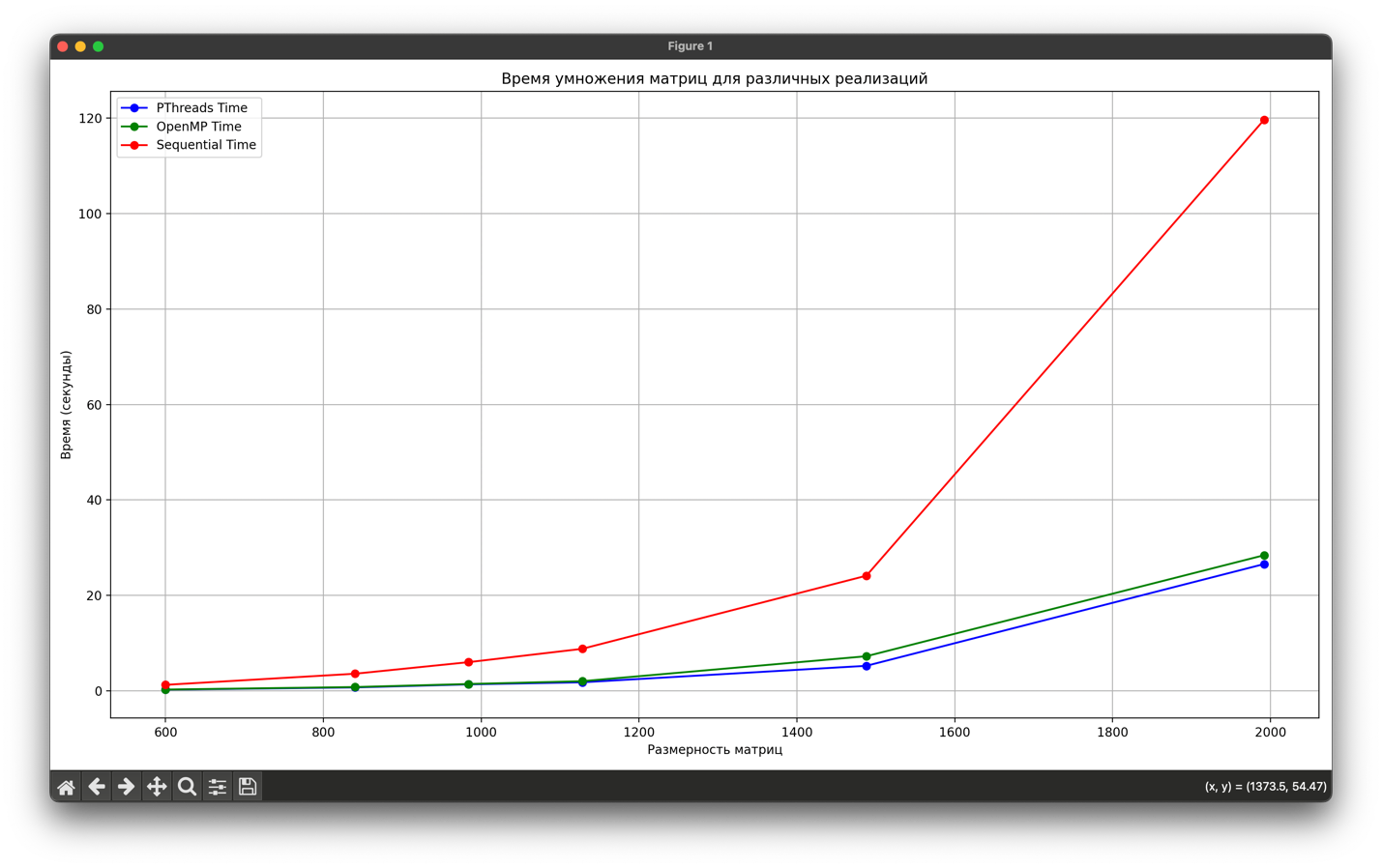


Рисунок 1 – Зависимость времени операции от размерности матрицы

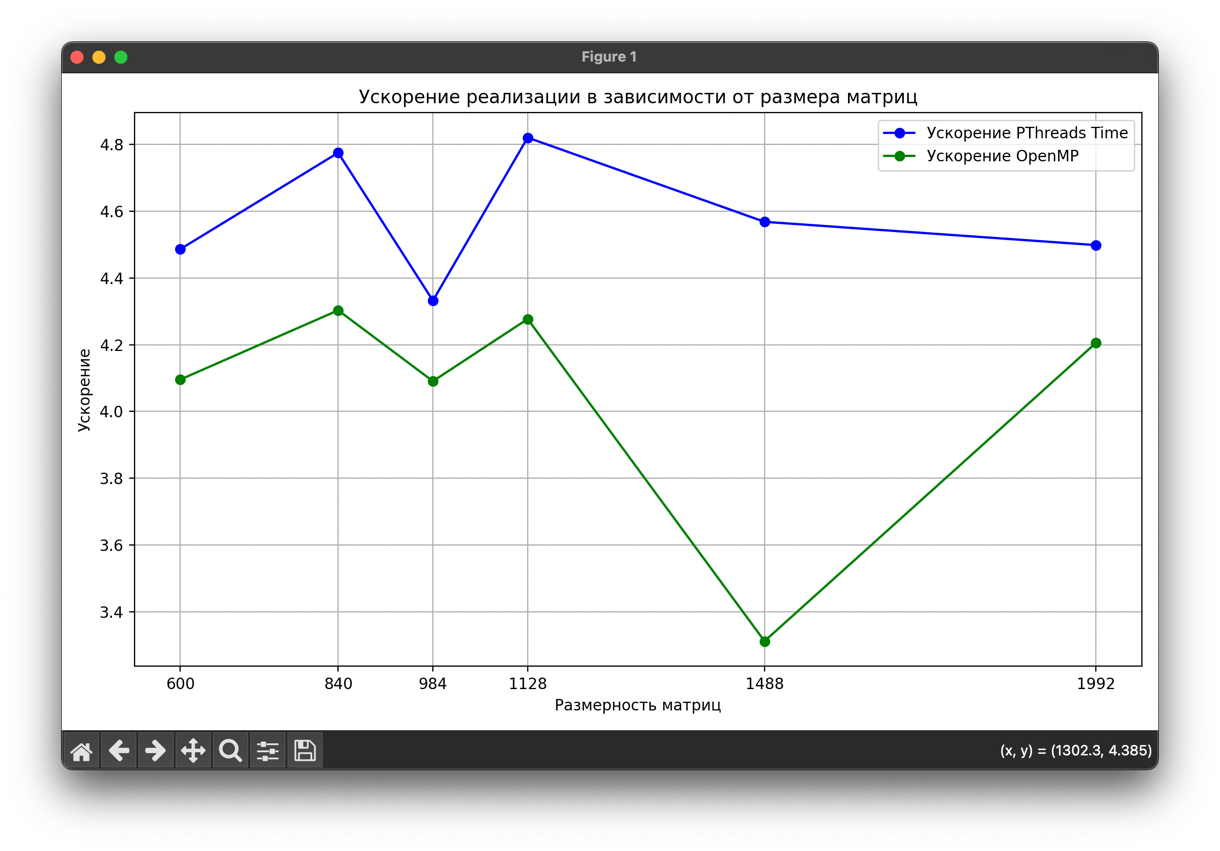


Рисунок 2 - Зависимость коэффициента ускореня от размерности матриц

Также были проведены эксперименты на процессоре Ryzen. Ниже предоставлены результаты работы программы. На рисунках 3-4 представлены зависимости времени и коэффициента ускорения от размерности матрицы.

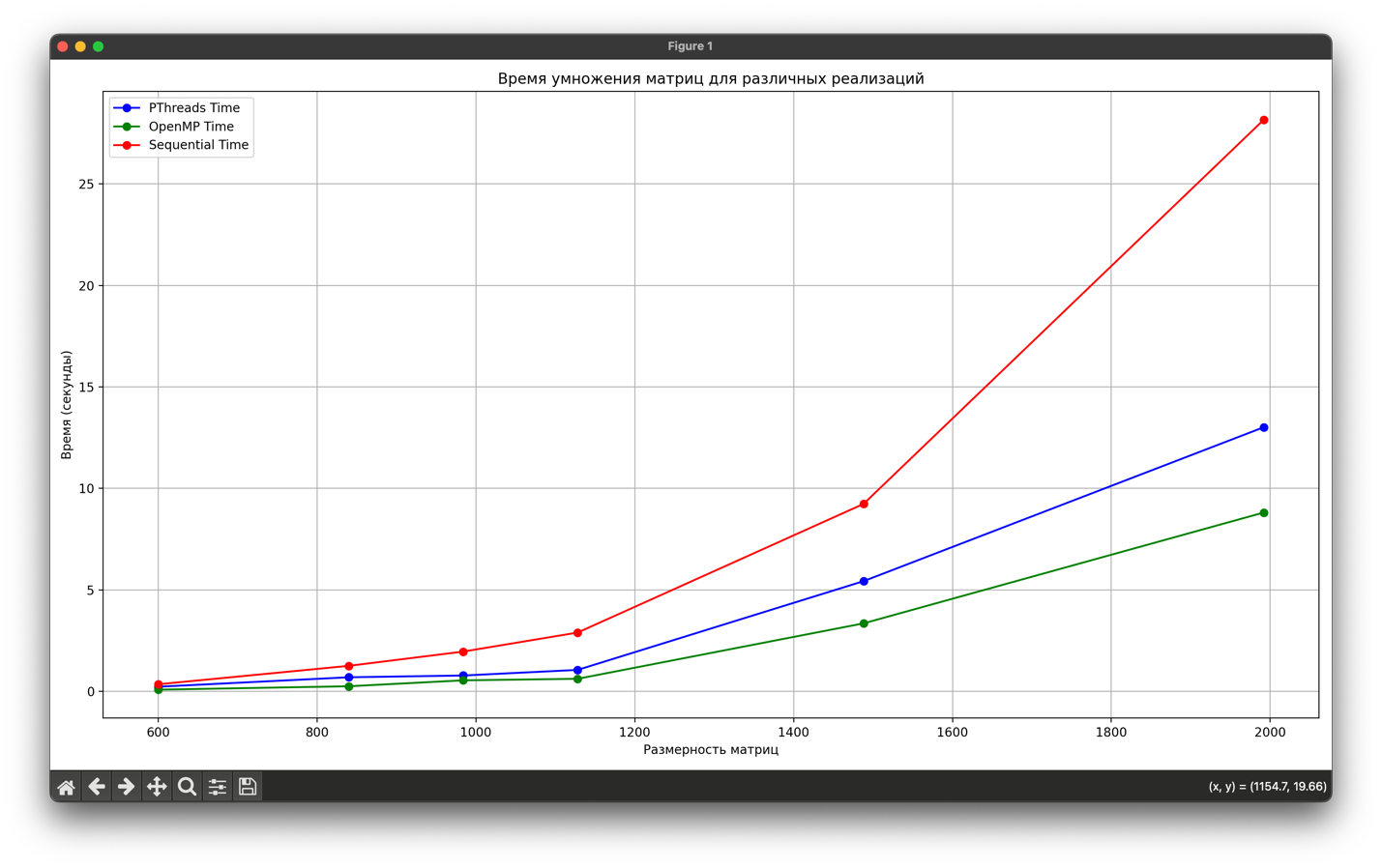


Рисунок 3 – Зависимость времени операции от размерности матрицы

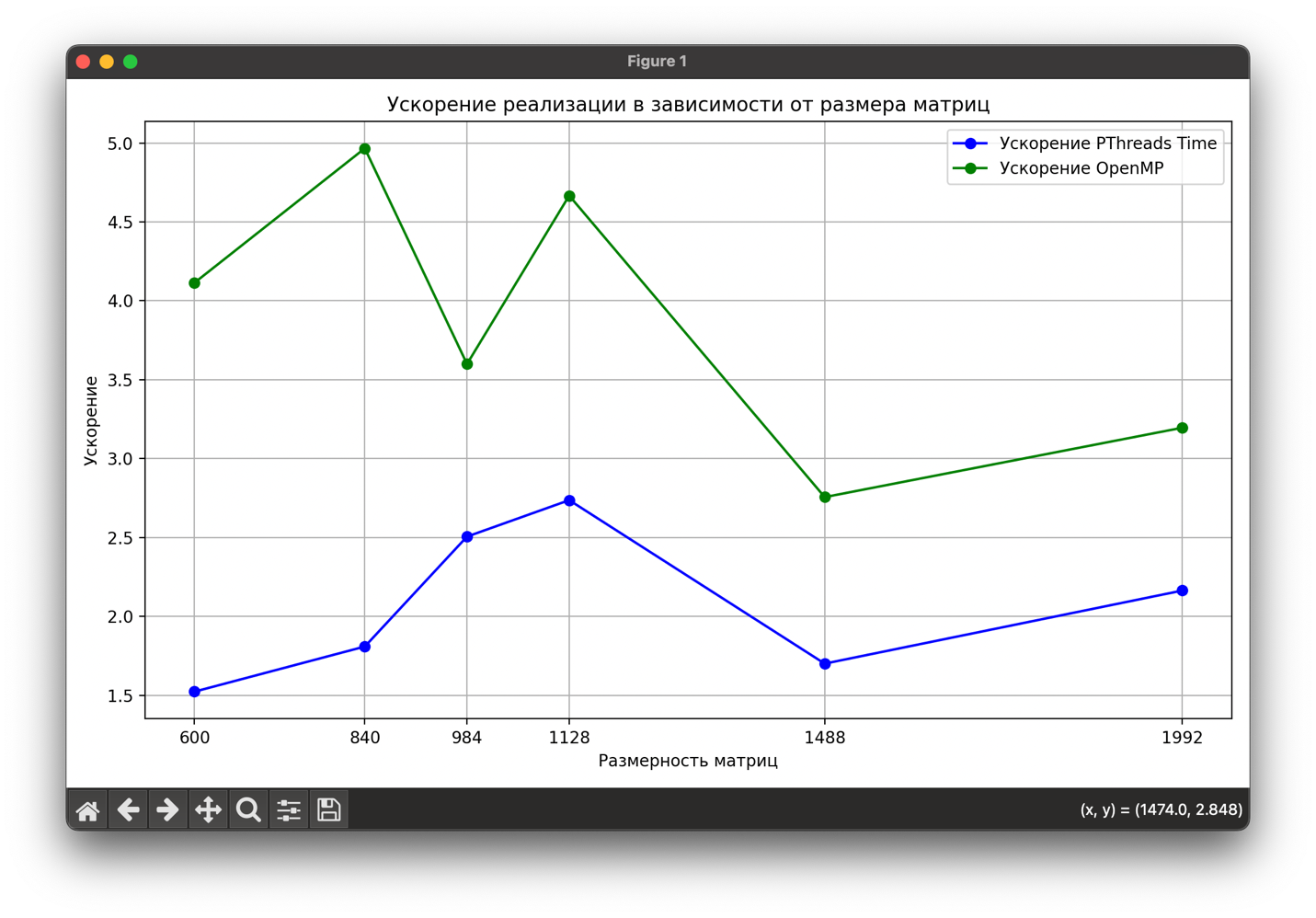


Рисунок 4 - Зависимость коэффициента ускореня от размерности матриц

Помимо этого, были проведены эксперименты на процессоре Intel. Ниже предоставлены результаты работы программы. На рисунках 5-6 представлены зависимости времени и коэффициента ускорения от размерности матрицы.

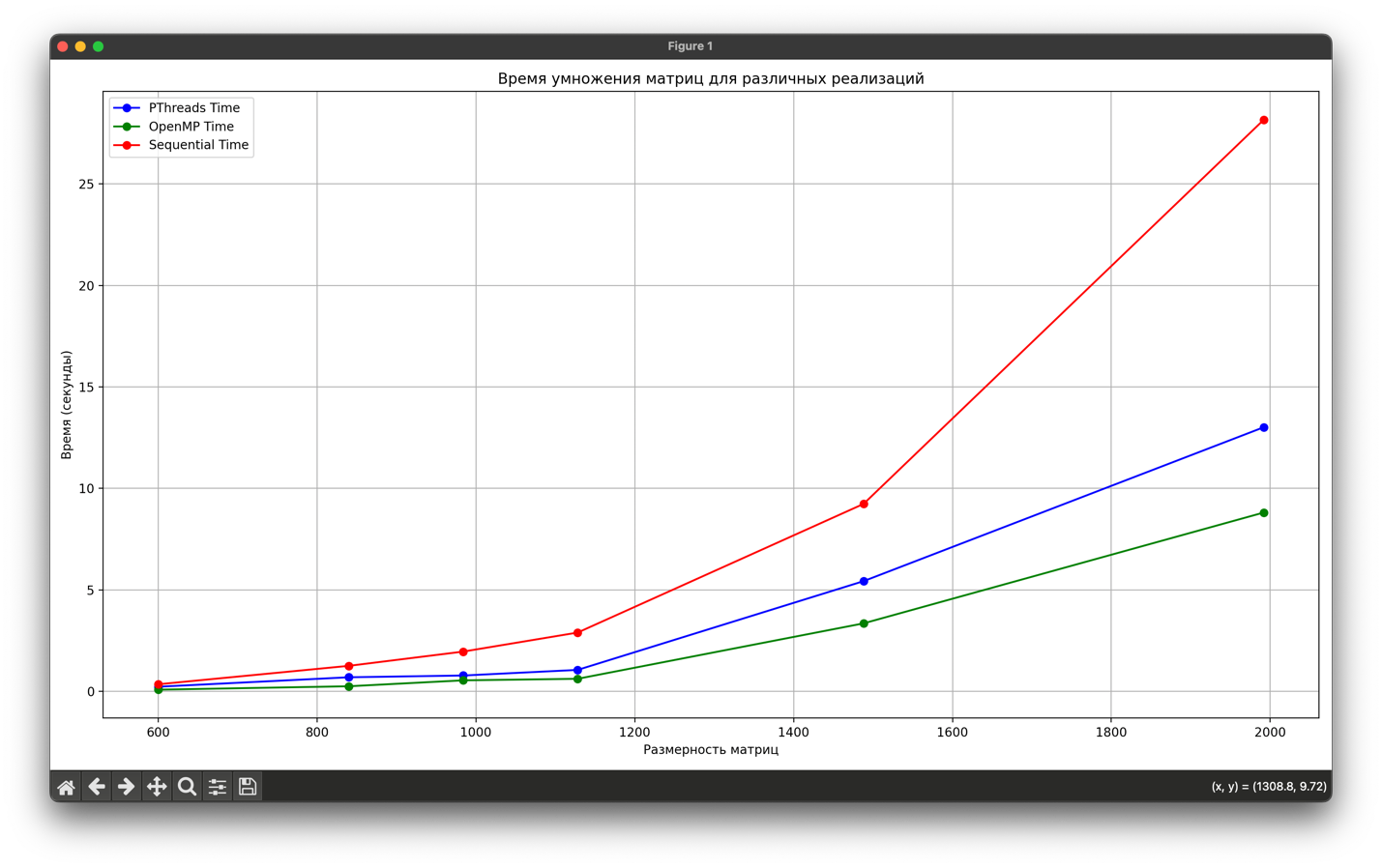


Рисунок 5 – Зависимость времени операции от размерности матрицы

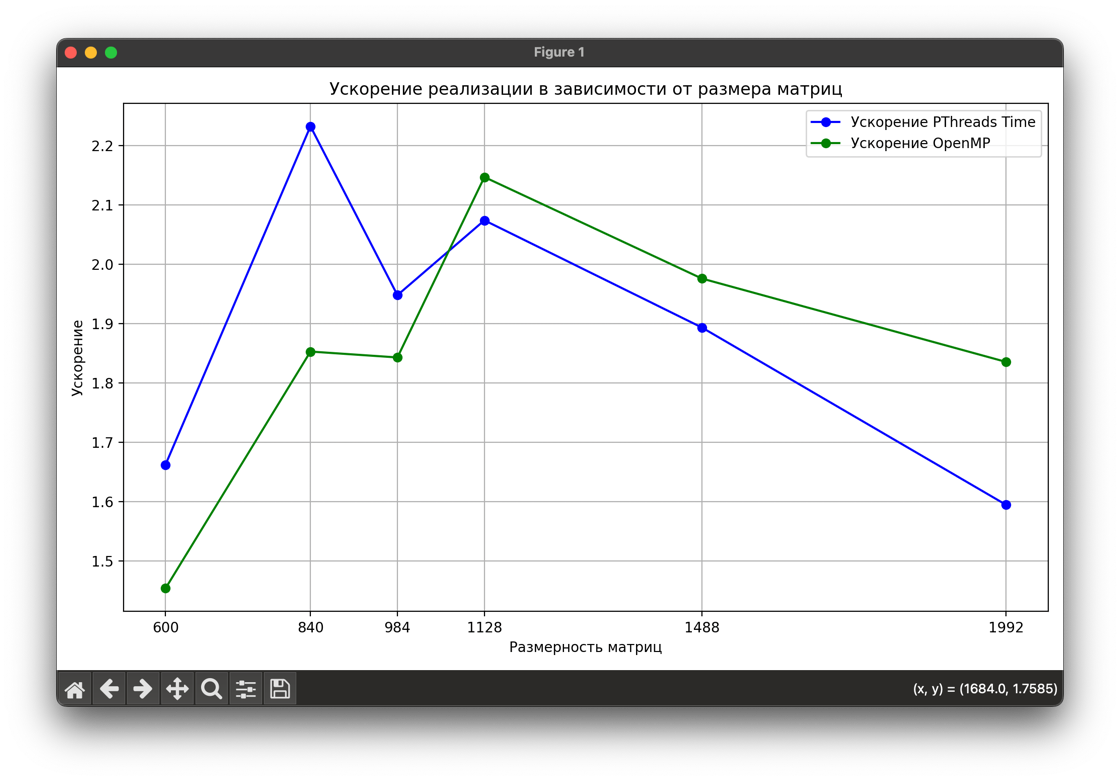


Рисунок 6 - Зависимость коэффициента ускореня от размерности матриц

Более подробную информацию зависимости времени операции от размерности матрицы и ускорения от размерности матрицы можно увидеть в таблицах, которые приведены ниже.

В таблице 1 предоставлен результат выполнения программы на процессоре M1-ARM64.

Таблица 1 – Результат выполнения программы на процессоре M1-ARM64 (в секундах).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матриц | Pthreads | OpenMP | Sequential | Boost PThreads | Boost OpenMP |
| 600 | 0.314761 | 0.432769 | 1.31538 | 4 | 3 |
| 840 | 0.759144 | 1.17173 | 3.64501 | 4 | 3 |
| 984 | 1.313 | 2.18777 | 6.07529 | 4 | 2 |
| 1128 | 1.84817 | 2.79948 | 8.8369 | 4 | 3 |
| 1488 | 5.30445 | 8.05342 | 24.4632 | 4 | 3 |
| 1992 | 25.6551 | 32.9846 | 122.823 | 4 | 3 |
| 3000 | 62.5539 | 92.5767 | 244.173 | 3 | 2 |

В таблице 2 предоставлен результат выполнения программы на процессоре Ryzen.

Таблица 2 – Результат выполнения программы на процессоре Ryzen (в секундах).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матриц | Windows Threads | OpenMP | Sequential | Boost WinThread | Boost OpenMP |
| 600 | 0.123105 | 0.0491584 | 0.233316 | 1 | 3 |
| 840 | 0.488179 | 0.218965 | 0.976427 | 3 | 5 |
| 984 | 0.501967 | 0.326972 | 1.496 | 2 | 5 |
| 1128 | 0.778647 | 0.529685 | 2.31558 | 2 | 4 |
| 1488 | 5.00661 | 2.81981 | 7.39594 | 1 | 2 |
| 1992 | 12.7988 | 7.77206 | 28.3054 | 2 | 3 |
| 3000 | 27.4976 | 26.3665 | 91.255 | 3 | 4 |

В таблице 3 предоставлен результат выполнения программы на процессоре Intel.

Таблица 3 – Результат выполнения программы на процессоре Intel (в секундах).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матриц | Windows Threads | OpenMP | Sequential | Boost WinThread | Boost OpenMP |
| 600 | 0.227506 | 0.117082 | 0.320119 | 1 | 2 |
| 840 | 0.781968 | 0.335842 | 1.1654 | 1 | 3 |
| 984 | 1.42835 | 0.682808 | 1.82857 | 1 | 2 |
| 1128 | 2.17557 | 1.14934 | 2.82978 | 1 | 2 |
| 1488 | 6.5746 | 3.67128 | 9.60693 | 1 | 2 |
| 1992 | 20.4062 | 13.4227 | 39.1359 | 1 | 2 |
| 3000 | 66.783 | 49.5544 | 151.92 | 2 | 3 |

ВЫВОД

По результатам работы приходим к выводу, что время работы трех реализаций на малых размерностях матрицы приблизительно одинаково. Однако, чем больше размерность матрицы, тем эффективность OpenMP и WinAPI сравнивается. В отличие от последовательного алгоритма, параллельные алгоритмы с использованием OpenMP и WinAPI работают значительно эффективнее, экономя достаточно много времени. Предпочтение выбора между OpenMP и WinAPI отдаётся, учитывая размерность матрицы и уровень сложности кода.

СПИСОК ЛИТЕРАТУРЫ

1. СТО 02068410-004-2018. Общие требования к учебным текстовым документам: методические указания [Электронный ресурс]. URL: https://ssau.ru/docs/sveden/localdocs/STO\_SGAU\_020684.. (дата обращения: 02.04.2022).
2. Воеводин В.В.. Воеводин Вл.В. Параллельные вычисления - СПб.: БХВ-Петербург. 2002. - 608 с. (дата обращения: 02.04.2022)
3. Эндрюс Г.Р. Основы многопоточного. параллельного и распределенного программирования [Электронный ресурс]. (дата обращения: 03.04.2022).
4. Корнеев В.В. Параллельные вычислительные системы - М.:Нолидж. 1999.– 320с. (дата обращения: 03.04.2022)
5. Жидченко В.В. Лекционный материал по курсу «Параллельное программирование» [Электронный ресурс]. URL: http://virtual6.ssau.ru/Moodle/course/view.php?id=973 (дата обращения: 03.04.2022).

ПРИЛОЖЕНИЕ A   
Листинг программы

#include <iostream>

#include <vector>

#include <pthread.h>

#include "/opt/homebrew/Cellar/libomp/18.1.8/include/omp.h"

#include <chrono>

#include <fstream>

void initialize\_matrices(int size, std::vector<std::vector<double> >& A, std::vector<std::vector<double> >& B) {

A.resize(size, std::vector<double>(size));

B.resize(size, std::vector<double>(size));

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

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

A[i][j] = i + j + 1;

B[i][j] = (i + j + 1) % 10 + 1;

}

}

}

struct ThreadData {

int start\_row;

int end\_row;

std::vector<std::vector<double> >\* A;

std::vector<std::vector<double> >\* B;

std::vector<std::vector<double> >\* C;

};

void\* multiply\_matrices\_pthreads(void\* arg) {

ThreadData\* data = static\_cast<ThreadData\*>(arg);

int start\_row = data->start\_row;

int end\_row = data->end\_row;

auto& A = \*(data->A);

auto& B = \*(data->B);

auto& C = \*(data->C);

for (int row = start\_row; row < end\_row; ++row) {

for (int j = 0; j < A.size(); ++j) {

C[row][j] = 0;

for (int k = 0; k < A.size(); ++k) {

C[row][j] += A[row][k] \* B[k][j];

}

}

}

return nullptr;

}

void multiply\_matrices\_openmp(std::vector<std::vector<double> >& C, const std::vector<std::vector<double> >& A, const std::vector<std::vector<double> >& B) {

#pragma omp parallel for

for (int i = 0; i < A.size(); ++i) {

for (int j = 0; j < A.size(); ++j) {

C[i][j] = 0;

for (int k = 0; k < A.size(); ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

}

void multiply\_matrices\_sequence(std::vector<std::vector<double> >& C, const std::vector<std::vector<double> >& A, const std::vector<std::vector<double> >& B) {

for (int i = 0; i < A.size(); ++i) {

for (int j = 0; j < A.size(); ++j) {

C[i][j] = 0;

for (int k = 0; k < A.size(); ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

}

void multiply\_matrices\_pthreads(int num\_threads, const std::vector<std::vector<double> >& A, const std::vector<std::vector<double> >& B, std::vector<std::vector<double> >& C) {

pthread\_t threads[num\_threads];

ThreadData thread\_data[num\_threads];

int rows\_per\_thread = A.size() / num\_threads;

for (int i = 0; i < num\_threads; ++i) {

thread\_data[i].start\_row = i \* rows\_per\_thread;

thread\_data[i].end\_row = (i + 1) \* rows\_per\_thread;

thread\_data[i].A = const\_cast<std::vector<std::vector<double> >\*>(&A);

thread\_data[i].B = const\_cast<std::vector<std::vector<double> >\*>(&B);

thread\_data[i].C = &C;

if (i == num\_threads - 1) {

thread\_data[i].end\_row = A.size();

}

pthread\_create(&threads[i], nullptr, multiply\_matrices\_pthreads, (void\*)&thread\_data[i]);

}

for (int i = 0; i < num\_threads; ++i) {

pthread\_join(threads[i], nullptr);

}

}

int main() {

std::vector<int> sizes;

sizes.push\_back(600);

sizes.push\_back(840);

sizes.push\_back(984);

sizes.push\_back(1128);

sizes.push\_back(1488);

sizes.push\_back(1992);

sizes.push\_back(3000);

const int num\_threads = 8;

std::ofstream output("results.csv");

output << "Size,Pthreads Time,OpenMP Time,Sequential Time\n";

for (int size : sizes) {

std::vector<std::vector<double> > A, B;

std::vector<std::vector<double> > C\_pthreads(size, std::vector<double>(size, 0));

std::vector<std::vector<double> > C\_openmp(size, std::vector<double>(size, 0));

std::vector<std::vector<double> > C\_sequence(size, std::vector<double>(size, 0));

initialize\_matrices(size, A, B);

auto start\_pthreads = std::chrono::high\_resolution\_clock::now();

multiply\_matrices\_pthreads(num\_threads, A, B, C\_pthreads);

auto end\_pthreads = std::chrono::high\_resolution\_clock::now();

auto duration\_pthreads = std::chrono::duration<double>(end\_pthreads - start\_pthreads).count();

auto start\_openmp = std::chrono::high\_resolution\_clock::now();

multiply\_matrices\_openmp(C\_openmp, A, B);

auto end\_openmp = std::chrono::high\_resolution\_clock::now();

auto duration\_openmp = std::chrono::duration<double>(end\_openmp - start\_openmp).count();

auto start\_sequence = std::chrono::high\_resolution\_clock::now();

multiply\_matrices\_sequence(C\_sequence, A, B);

auto end\_sequence = std::chrono::high\_resolution\_clock::now();

auto duration\_sequence = std::chrono::duration<double>(end\_sequence - start\_sequence).count();

output << size << "," << duration\_pthreads << "," << duration\_openmp << "," << duration\_sequence << "\n";

std::cout << "Size: " << size

<< " | PThreads: " << duration\_pthreads

<< " | OpenMP: " << duration\_openmp

<< " | Sequential: " << duration\_sequence << "\n";

}

output.close();

return 0;

}