МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)   
  
  
Факультет информатики  
Кафедра программных систем  
  
Дисциплина  
**Параллельное программирование  
  
  
  
ОТЧЕТ**по лабораторной работе №1

«Параллельные вычисления в разделяемой памяти»

Студенты: Садовников К.А.

Колбанов Д.О.

Атякшев С.А.  
Группа: 6401-020302D   
  
Преподаватель: Востокин С.В.

Самара 2024

**Задание 1.** Требуется реализовать параллельный алгоритм построчного вычисления произведения квадратных матриц в разделяемой памяти. Сравнить производительность OpenMP-реализации и реализации, использующей API операционных систем.

**Листинг программы.**

#include <iostream>

#include <omp.h>

#include <ctime>

#include <cstdlib>

#include <vector>

using namespace std;

int main()

{

setlocale(LC\_CTYPE, "Russian");

double maxBorder = INT32\_MAX;

double minBorder = -maxBorder;

srand(time(NULL));

double vedro = rand() \* (maxBorder - minBorder) / RAND\_MAX + minBorder;

for (int n = 5; n <= 2000; n+=50) {

vector <vector<double>> first(n, vector<double>(n));

vector <vector<double>> second(n, vector<double>(n));

vector <vector<double>> result(n, vector<double>(n));

cout << "Размерность матриц - " << n << endl;

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

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

first[i][j] = rand() \* (maxBorder - minBorder) / RAND\_MAX + minBorder;

second[i][j] = rand() \* (maxBorder - minBorder) / RAND\_MAX + minBorder;

}

}

double timer = omp\_get\_wtime();

#pragma omp parallel for shared(first, second, result)

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

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

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

result[i][j] += first[i][k] \* second[k][j];

}

}

}

cout << "\t" << "Время с распараллеливанием - " << omp\_get\_wtime() - timer << endl;

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

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

result[i][j] = 0;

}

}

timer = omp\_get\_wtime();

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

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

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

result[i][j] += first[i][k] \* second[k][j];

}

}

}

cout <<"\t" << "Время без распараллеливания - " << omp\_get\_wtime() - timer << endl <<endl;

}

}

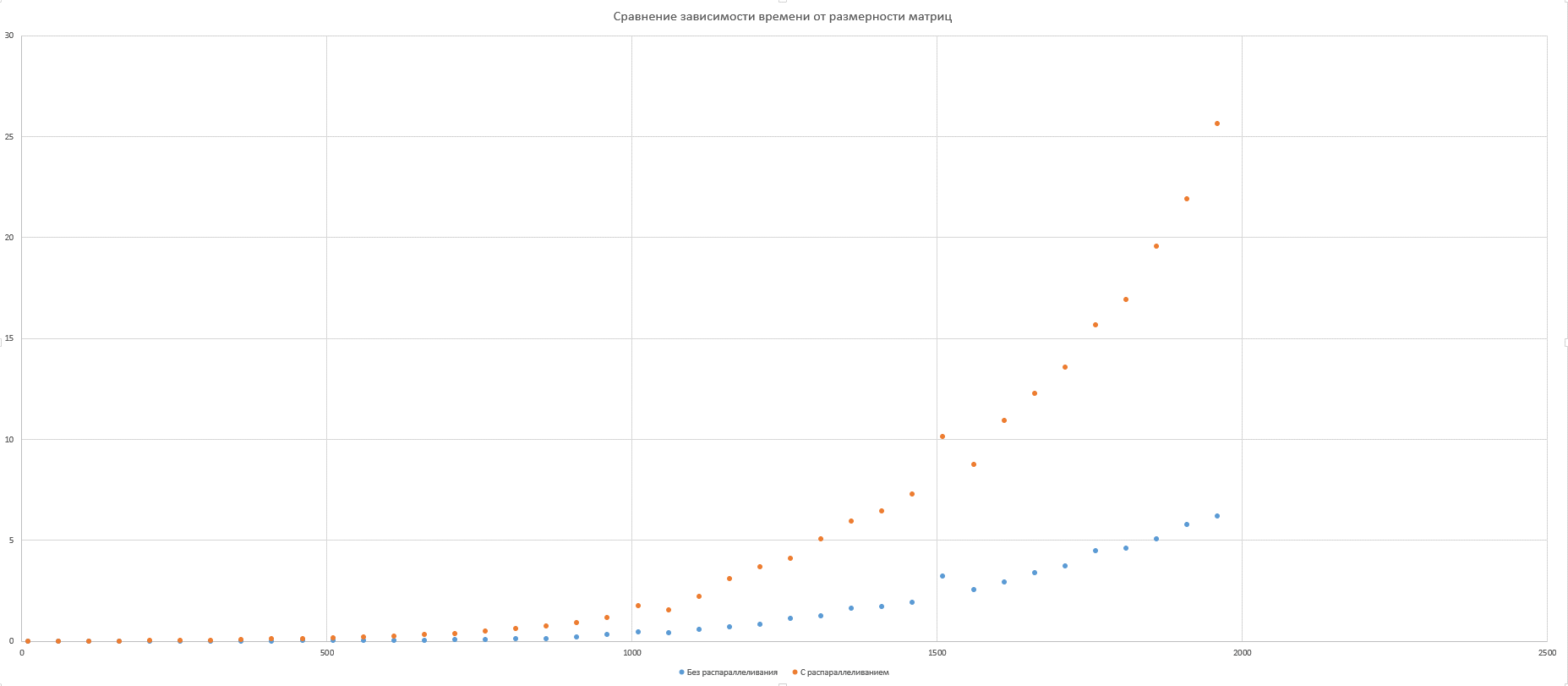
Результаты выполнения программы изображены на рисунке 1. Ось y – время в секундах, а ось x – размерность матриц.

Рисунок 1 – Сравнение зависимости времени от размерности матриц

Таким образом, мы наблюдаем, что при размерности матриц начиная с 500 распараллеливание начинает оказывать заметное влияние.

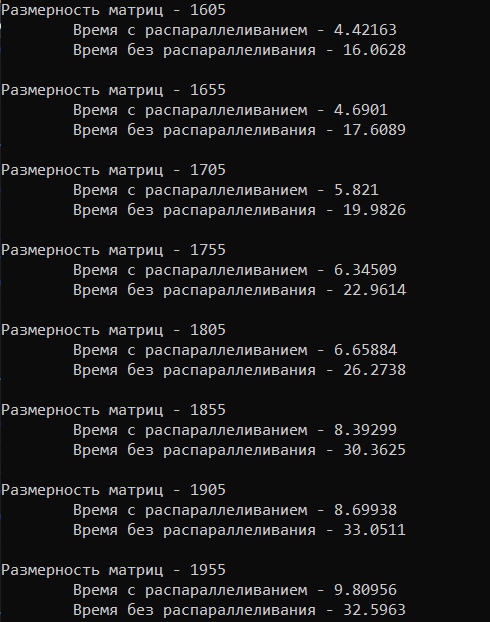


Рисунок 2 – Сравнение зависимости времени от размерности матриц при большой размерности матриц на втором компьютере

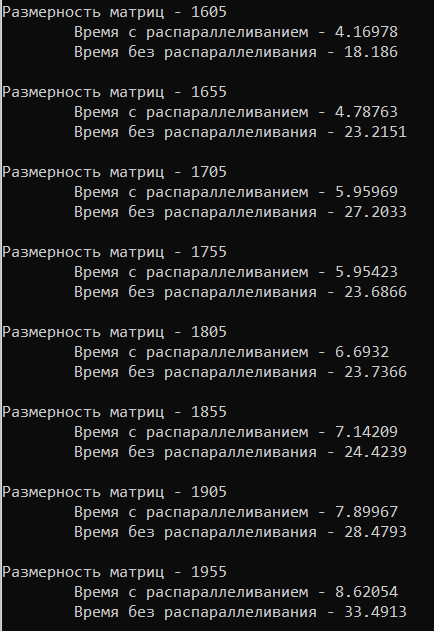


Рисунок 3 – Сравнение зависимости времени от размерности матриц при большой размерности матриц на третьем компьютере

Однако при более тщательном рассмотрении перемножения при маленьких размерностях матриц наблюдается обратная картина.

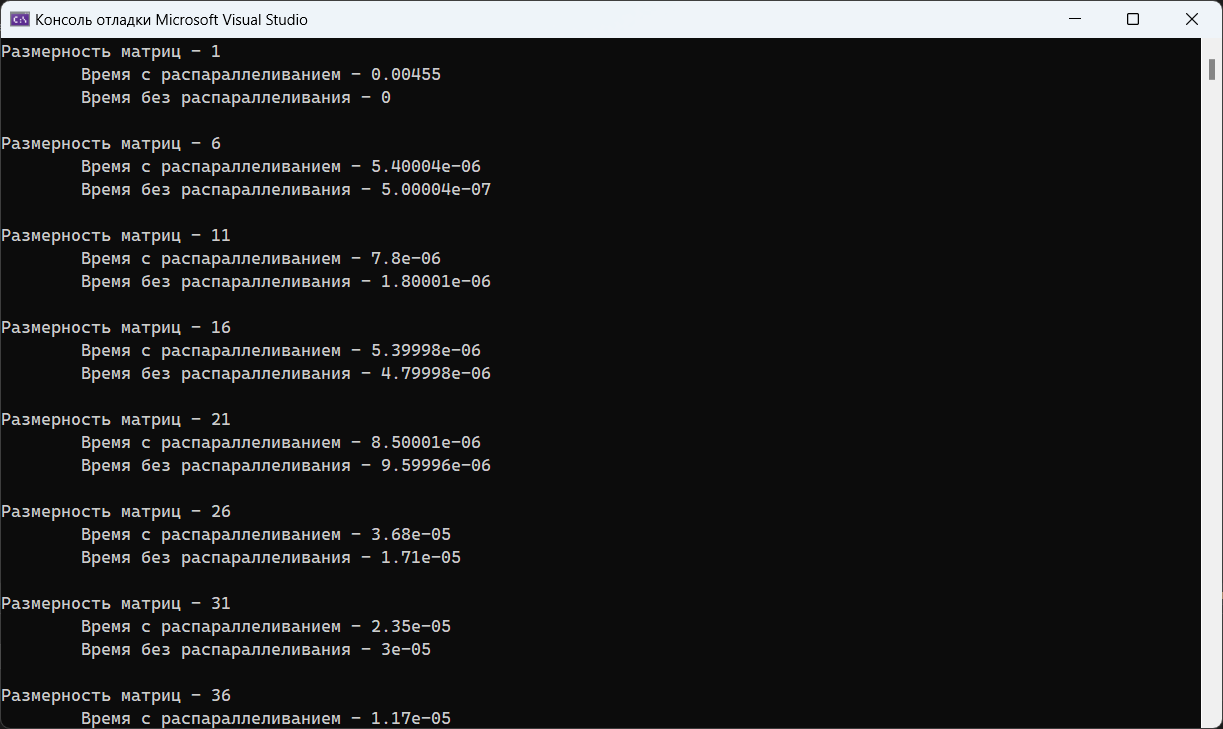


Рисунок 2 – Сравнение зависимости времени от размерности матриц при небольшой размерности матриц на первом компьютере

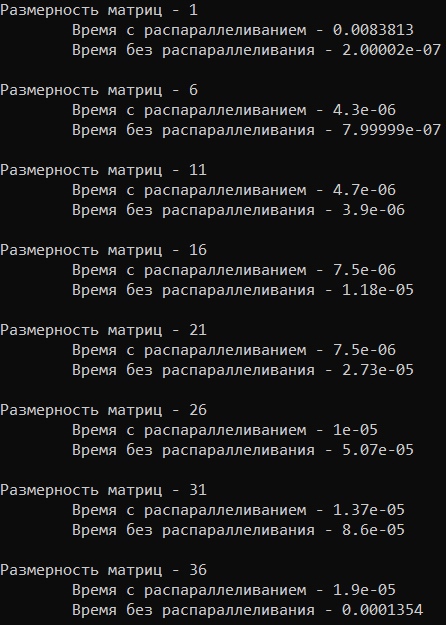


Рисунок 3 – Сравнение зависимости времени от размерности матриц при небольшой размерности матриц на втором компьютере

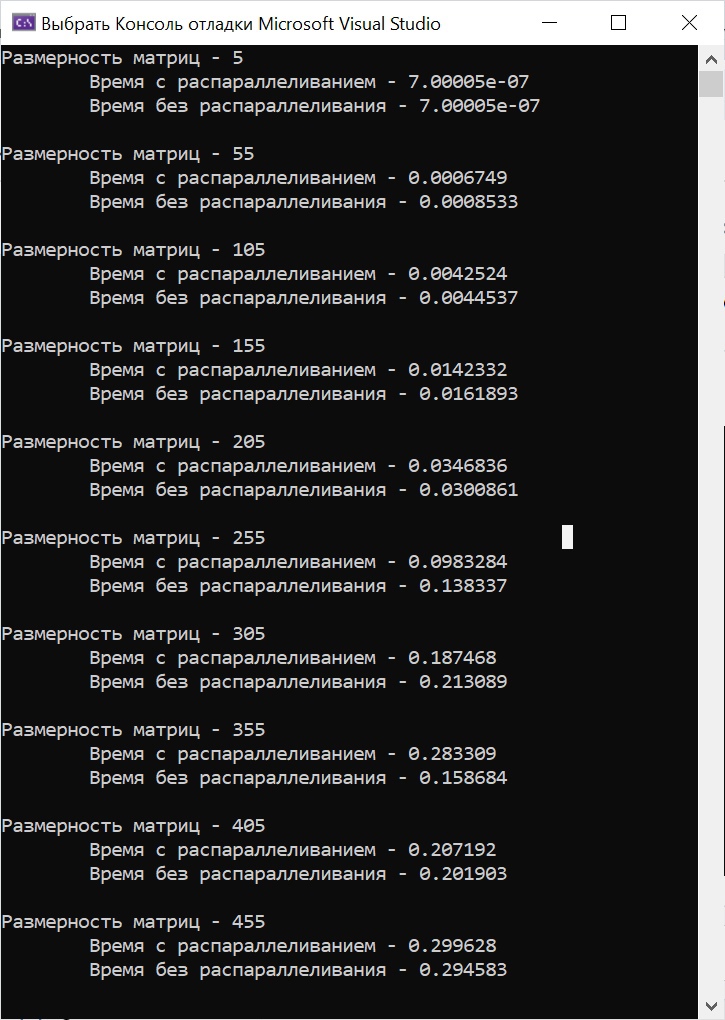


Рисунок 4 – Сравнение зависимости времени от размерности матриц при небольшой размерности матриц на третьем компьютере

При маленьких размерах матриц нецелесообразно использовать параллельные вычисления, так как временные затраты на само распараллеливание сделают их неэффективными.