|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
|  | | | |
| Расчетно-графическое задание № 1 | | | |
| по дисциплине «Архитектура ЭВМ и ВС» | | | |
| **Определение времени работы прикладных программ** | | | |
|  | | | |
|  | Факультет: | ПМ |
| Группа: | ПМ-21 |
| Студент: | Егупов иван |
|  |  |
| Преподаватель | щукин георгий анатольевич |
| Дата | 19.12.2023 |
|  |  |
|  | | | |
| Новосибирск | | | |

**1. Цель работы**

1. Изучить методы измерения времени работы подпрограммы.
2. Научиться повышать точность измерения времени работы подпрограмм.
3. На практике воспользоваться полученными знаниями.

**2. Текст программы**

#include <iostream>

#include <locale.h>

#include <random>

#include <ctime>

using namespace std;

clock\_t Mult(int\*\* Mat1, int\*\* Mat2, int\*\* Res, int N);

clock\_t Mult1(int\*\* Mat1, int\*\* Mat2, int\*\* Res, int N);

void transpose(int\*\* Mat2, int N);

int main()

{

setlocale(0, "");

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dist(-100, 100);

int N;

cout << "Введите размерность матриц: ";

cin >> N;

int \*\*a = new int\* [N];

int \*\*b = new int\* [N];

int \*\*c = new int\* [N];

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

{

a[i] = new int[N];

b[i] = new int[N];

c[i] = new int[N];

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

{

a[i][j] = dist(gen);

b[i][j] = dist(gen);

c[i][j] = 0;

}

}

cout << "Затраченное время при стандартном обходе: " << (float)Mult(a,b,c,N) / CLOCKS\_PER\_SEC << " с." << endl;

transpose(b, N);

cout << "Затраченное время при последовательном обходе: " << (float)Mult1(a,b,c,N) / CLOCKS\_PER\_SEC << " с." << endl;

return 0;

}

clock\_t Mult(int\*\* Mat1, int\*\* Mat2, int\*\* Res, int N) // Умножение со стандартным обходом

{

clock\_t start = clock();

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

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

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

Res[i][j] += Mat1[i][k] \* Mat2[k][j];

clock\_t end = clock();

return end - start;

}

clock\_t Mult1(int\*\* Mat1, int\*\* Mat2, int\*\* Res, int N) //Умножение с последовательным обходом данных

{

clock\_t start = clock();

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

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

Res[i][j] = 0;

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

Res[i][j] += Mat1[i][k] \* Mat2[j][k];

}

clock\_t end = clock();

return end - start;

}

void transpose(int\*\* Mat2, int N)

{

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

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

int temp;

temp = Mat2[i][j];

Mat2[i][j] = Mat2[j][i];

Mat2[j][i] = temp;

}

}

**3. Графики зависимости времени счета от размера матриц**

Обстоятельства тестирования:

|  |  |
| --- | --- |
|  | **Домашний компьютер** |
| **Версия компилятора Visual Studio** | Версия 17.5.5 |
| **Операционная система** | Windows 11 Home x64 |
| Версия 22H2 |
| Сборка 22621.2861 |
| **Процессор** | AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz |
| 6 ядра |
| 12 логических процессора |
| **Кэш** | L1 – 384 КБ  L2 – 3,0 МБ  L3 – 8,0 МБ |
| **ОЗУ** | 16,00 ГБ |

Тестирование:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность матрицы | Od, c | | O1, c | | Ox, c | |
| Без транспонирования / С транспонированием | | | | | |
| 500 | 0.444 | 0.420 | 0.222 | 0.226 | 0.220 | 0.229 |
| 700 | 1.392 | 1.155 | 0.692 | 0.635 | 0.748 | 0.680 |
| 900 | 3.860 | 2.458 | 1.888 | 1.377 | 1.658 | 1.387 |
| 1100 | 7.993 | 4.566 | 3.653 | 2.524 | 3.669 | 2.522 |
| 1300 | 14.117 | 7.394 | 6.305 | 4.148 | 6.319 | 4.142 |
| 1500 | 20.238 | 11.303 | 9.254 | 6.258 | 9.289 | 6.295 |
| 1700 | 32.283 | 16.653 | 14.720 | 9.134 | 14.643 | 9.136 |
| 1900 | 46.733 | 22.808 | 23.256 | 12.699 | 23.378 | 12.764 |

1. **Выводы по результатам работы**

При выполнении расчётно-графического задания, мы написали программу умножения двух квадратных матриц, которая работала по принципу: обход первой матрицы происходит по строкам, умножается на вторую матрицу, обход которой происходит по столбцам. Затем мы модифицировали программу, и она стала работать немного по другому принципу: матрицу, обход которой при умножении происходит по столбцам, транспонируется до умножения, то есть при умножении элементы обеих матриц считываются последовательно, по строкам.

Мы изучили методы измерения времени работы подпрограммы. Научились повышать точность измерения времени работы подпрограмм. Провели работу программ на разных уровнях оптимизации компилятора. На практике воспользовались полученными знаниями.

Также, мы узнали, что для перемножения матриц лучше использовать второй алгоритм, так как он эффективнее, в том числе и на разных уровнях оптимизации, что видно на графиках. Неэффективность первого алгоритма обусловлено тем, что чтение матрицы, на которую производилось умножение из оперативной памяти, осуществлялось по столбцам. Также, умножение на транспонированную матрицу происходит быстрее, потому что процессор уже положил матрицу в кэш и ему не нужно идти за ней в оперативную память.

Сравнивая результаты работы программы на уровнях оптимизации, можно сказать, что Release х86 (/Ox) и Release х64 (/O1) примерно в два раза эффективнее по времени выполнения программы, чем Debug х86 (/Od)