МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Инженерная школа информационных технологий и робототехники Направление: Информатика и вычислительная техника

Отделение информационных технологий

Отчет по лабораторной работе №2 по дисциплине

«Организация ЭВМ»

«ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ SIMD-ИНСТРУКЦИЙ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА»

Вариант 28

Выполнил: Студент группы 8В01

И.П. Тюрин

Проверил: Инженер ОИТ

##### Ю.В. Котова

Томск 2022

**Цель работы**

Изучить понятие SIMD (Single Instruction Multiple Data). Рассмотреть принцип работы SIMD-инструкций процессора на примере программы, выполняющей обработку однотипных данных.

**Задание**

Разработать программу на языке C++ для работы с SIMD-инструкциями процессора. Выполнить следующие пункты:

1. Сгенерировать 2 матрицы (A и B), размерностью 4096 x 4096, состоящие из произвольных чисел заданного (согласно варианту) типа.
2. Вычислить произведение матриц с использованием SIMD-инструкций согласно варианту задания (таблица 1) и с использованием скалярных вычислений.
3. Сравнить быстродействие векторной и скалярной реализаций. Сделать выводы.
4. Убедиться, что векторное умножение работает правильно путём сравнения с результатами скалярного умножения.
5. Рассчитать размер памяти, требуемый для хранения матриц. Сравнить рассчитанный размер с памятью, выделенной под программу в диспетчере задач. Сделать выводы.

Задание можно выполнять, используя любой компилятор, поддерживающий SIMD-инструкции, и в любой операционной системе.

**Ход работы**



Рис. 1.Вариант задания

**Код программы**

#include <iostream>

#include <immintrin.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

const unsigned int msize = 4096;

void scalar\_mul(int32\_t\*\* A, int32\_t\*\* B, int32\_t\*\* C, unsigned int N) {

cout << "Scalar multiplication:" << endl;

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

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

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

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

}

}

}

}

int vector\_mul(int32\_t\*\* A, int32\_t\*\* TB,int32\_t\*\* C, unsigned int N) {

cout << "SSE multiplication:" << endl;

if (N < 4) return 0;

int32\_t row[4] = { 0 };

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

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

for (int k = 0; k < N; k += 4) {

\_\_m128i c\_line = \_mm\_setzero\_si128(); // заполняем результирующий SSE-регистр нулями

\_\_m128i a\_line = \_mm\_load\_si128((\_\_m128i\*) & A[i][k]); // загружаем 4 элемента int из массива A в SSE-регистр

\_\_m128i b\_line = \_mm\_load\_si128((\_\_m128i\*) & TB[j][k]); // загружаем 4 элемента int из массива B в SSE-регистр

c\_line = \_mm\_mullo\_epi32(a\_line, b\_line); // Умножение двух SSE-регистров

\_mm\_store\_si128((\_\_m128i\*) &row, c\_line);

C[i][j] += row[0] + row[1] + row[2] + row[3];

}

}

//cout << "and anoter one" << endl;

}

}

string matrix\_comparing(int32\_t\*\* A, int32\_t\*\* B, unsigned int N) {

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

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

if (A[i][j] != B[i][j]) {

return "false";

}

}

}

return "true";

}

void show\_m(int32\_t\*\* m, unsigned int N) {

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

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

cout << m[i][j] << " ";

}

cout << endl;

}

}

int main()

{

srand(time(NULL));

cout << "Initialization" << endl;

double overall = 0.0;

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

alignas(16) int32\_t \*\*A = new int32\_t\*[msize];

for (int32\_t i = 0; i < msize; i++) {

A[i] = new int32\_t[msize];

}

alignas(16) int32\_t \*\*B = new int32\_t\*[msize];

for (int32\_t i = 0; i < msize; i++) {

B[i] = new int32\_t[msize];

}

alignas(16) int32\_t \*\*C\_scalar = new int32\_t\*[msize];

for (int32\_t i = 0; i < msize; i++) {

C\_scalar[i] = new int32\_t[msize];

}

alignas(16) int32\_t\*\* C\_vector = new int32\_t \* [msize];

for (int32\_t i = 0; i < msize; i++) {

C\_vector[i] = new int32\_t[msize];

for (int32\_t j = 0; j < msize; j++) {

A[i][j] = (uint32\_t)rand() % 100;

B[i][j] = (uint32\_t)rand() % 100;

C\_scalar[i][j] = 0;

C\_vector[i][j] = 0;

}

}

alignas(16) int32\_t\*\* TB = new int32\_t \* [msize];

for (int32\_t i = 0; i < msize; i++) {

TB[i] = new int32\_t[msize];

for (int32\_t j = 0; j < msize; j++) {

TB[i][j] = B[j][i];

}

}

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

auto duration = duration\_cast<microseconds>(end - start);

cout << "time : " << duration.count() << endl;

overall = 0.0;

start = high\_resolution\_clock::now();

scalar\_mul(A, B, C\_scalar, msize);

end = high\_resolution\_clock::now();

duration = duration\_cast<microseconds>(end - start);

cout << "time : " << duration.count() << endl;

overall = 0.0;

start = high\_resolution\_clock::now();

vector\_mul(A, TB, C\_vector, msize);

end = high\_resolution\_clock::now();

duration = duration\_cast<microseconds>(end - start);

cout << "time : " << duration.count() << endl;

/\*

show\_m(C\_scalar, msize);

cout << endl;

show\_m(C\_vector, msize);

cout << endl;

\*/

cout << "matrix equality : " << matrix\_comparing(C\_scalar, C\_vector, msize) << endl;

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

delete[] A[i];

}

delete[] A;

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

delete[] B[i];

}

delete[] B;

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

delete[] C\_scalar[i];

}

delete[] C\_scalar;

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

delete[] C\_vector[i];

}

delete[] C\_vector;

return 0;

}

**Результат работы программы**

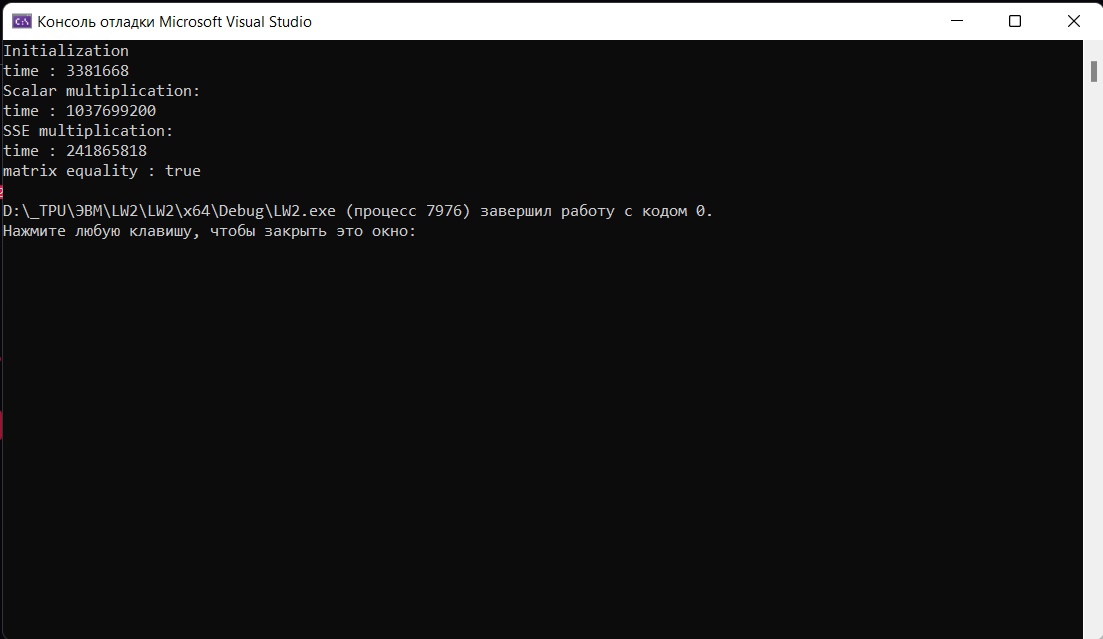


Рис. 1. Вычисление произведения матриц с использованием SIMD инструкций и без

Размер двумерной матрицы с целочисленными 32-битными значениями 4096 x 4096 равен 64 Мбайт. В программе хранится 5 таких матриц, следовательно занимать они будут 320 Мбайт

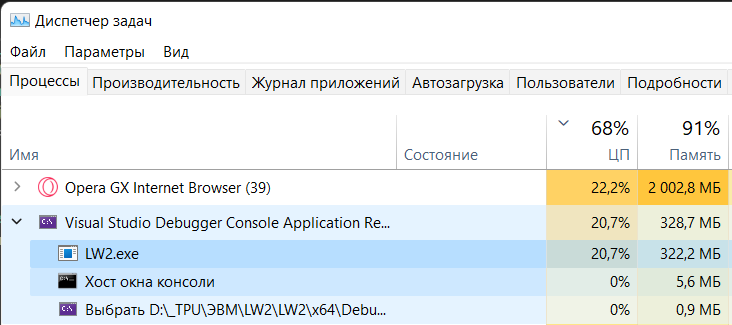


Рис. 2. Память занимаемая программой примерно равна рассчитанной (Оставшиеся 2.2 Мбайта занимает остальная программа)

**Вывод**

В ходе лабораторной работы была создана программа, вычисляющая произведение двух матриц размерностью 4096 x 4096 скалярным методом и с помощью SIMD инструкций. Также было вычислено время выполнения обоими способами. Вычисление с помощью SIMD инструкций оказалось примерно в 4 раза быстрее.