Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе № 3
Измерение скорости вычисления операции “GAXPY” на языке программирования C++ с различным состоянием инструкций»

Курс: «Параллельные алгоритмы»

Студент: Грязнов Илья Евгеньевич
(фамилия, имя отчество)
Группы 6131-010402D

Самара 2022

Содержание

Задание

В целях ознакомления с инструкциями SSE, приобретения навыков программирования параллельных вычислений и получения знаний о изменении производительности программы при выполнении с различным набором инструкций для вычислений произвести расчеты, с помощью уже реализованной в программном виде операции “gaxpy” на языке программирования C++, с различными значениями матриц, подаваемых на вход и замерить скорость выполнения программы. Записать входные и выходные данные.

Теоретическая часть

Умножение матриц является одной из базовых операций вычислительной линейной алгебры. К ней неизменно сводятся блочные алгоритмы матричных разложений, которые в свою очередь широко применяются при решении систем линейных алгебраических уравнений и алгебраической проблемы собственных значений – двух основных задач упомянутой предметной области.

Операцию gaxpy (general A x plus y.) принято записывать в виде

$$z=Ax+y$$

Через нее удобно выражать умножение матриц, матричные разложения и итерационные методы решения СЛАУ.

Положим, что вектора и матрица из gaxpy для удобства составления параллельного алгоритма разбиты на блоки: $z_i, y_i \in \mathbb{R}^{\alpha \times 1}$, где $1 \leq i, j \leq p$, $\alpha = n/p$, $\beta = m/p$, а p – количество задач искомого алгоритма. Построение параллельного алгоритма начинается с распределения блоков матрицы A между задачами; Выбранный способ распределения служит для дальнейшей классификации алгоритмов.

SSE (Streaming SIMD Extensions, потоковое SIMD-расширение процессора) - это SIMD (Single Instruction, Multiple Data, Одна инструкция - множество данных) набор инструкций, разработанный Intel и впервые представленный в процессорах серии Pentium III как ответ на аналогичный набор инструкций 3DNow! от AMD, который был представлен годом раньше. Первоначально названием этих инструкций было KNI — Katmai New Instructions (Katmai - название первой версии ядра процессора Pentium III).

Технология SSE позволяла преодолеть две основные проблемы MMX: при использовании MMX невозможно было одновременно использовать инструкции сопроцессора, так как его регистры были общими с регистрами MMX, и возможность MMX работать только с целыми числами.

SSE включает в архитектуру процессора восемь 128-битных регистров и набор инструкций, работающих со скалярными и упакованными типами данных.

Преимущество в производительности достигается в том случае, когда необходимо произвести одну и ту же последовательность действий над разными данными. В таком случае блоком SSE осуществляется распараллеливание вычислительного процесса между данными.

Ход выполнения работы

Взяв за основу реализованный код программы на C++ в предыдущей лабораторной работе, отвечающий за создание квадратной матрицы заданного размера и двух векторов, длинной соответственно подходящей для выполнения операции gaxpy, дополним его функций измерения скорости выполнения программы и уберем более не нужные нам записи данных о матрице, векторах и результатах вычисления в файл. Далее заменим в программе, непосредственно выполняющей вычислительные операции для данных объектов с числовыми значениями типа double (число с плавающей точкой,

размер в байтах - 8), на тип данных float (размер в байтах - 4). Сама реализация программы представлена ниже в приложении (после списка используемой литературы).

После того, как произвели необходимые изменения в коде программы, запускаем ее работу с различными размерами матрицы на входе и замеряем скорость ее работы при отключенных расширенных инструкций SIMD (/arch:SSE) и затем со включенными. Работа производилась в IDE Visual Studio 2022 Community Edition. Все изменения параметров при компиляции можно производить в настройках программы для различных конфигураций запуска, таких как: Debug или Release, во вкладке «проект» -> «свойства»:

Полученные в консольном приложении, написанном на C++ данные записываем в txt файл. Все результаты представлены в таблице ниже данного отчета.

Данные полученные при выполнении программы:

Размерность матрицы	Время выполнения без SSE	Время выполнения с SSE
10x10	0,023674s	0,058826s
100x100	0,559215s	0,592560s
200x200	2,188558s	1,865936s
300x300	8,181087s	1,710880s
400x400	35,575073s	2,200071
500x500	95,899536s	2,179954s
600x600	210,057159s	2,448652s
700x700	524,537842s	2,915661s
800x800	-	2,946222s
900x900	-	3,093331s
1000x1000	-	3,430650s
10000x10000	-	25,749996s

Таблица 1. Результаты работы операции gaxpy с различными параметрами компиляции

*Прочерками в таблице обозначены отсутствующие данные, ввиду длительности выполнения операции.

Вывод

В рамках данной лабораторной работы по изучению параллельных алгоритмов и инструкций для компиляции SSE была изменена ранее реализованная операция gaxpy на языке C++. Производилась компиляция данной программы с включёнными инструкциями SSE и без, для различных входных данных (размеров матриц). Время работы программы записывалось для дальнейшего анализа.

Результаты весьма наглядно отображают полезность работы инструкций SSE, заметно и очень ощутимо увеличилась скорость вычислений. Это было заметно еще до получения выходных данных, программа отрабатывала в разы быстрее при включённых инструкциях SSE при компиляции.

Запуск программы с размерами матриц более 700x700 и отключенными инструкциями SSE при компиляции не производился, ввиду заметно возросшего времени выполнения программы.

Список использованных источников

1. Головашкин Д.Л. Векторные алгоритмы вычислительной линейной алгебры: учебной пособие – Самара: Изд-во Самарского университета, 2019. – 76 с.
2. Головашкин Д.Л. Параллельные алгоритмы вычислительной линейной алгебры: учебной пособие – Самара: Изд-во Самарского университета, 2019. – 88 с.
3. Головашкин Д.Л. Модели в теории параллельных вычислений: учебной пособие – Самара: Изд-во Самарского университета, 2019. – 96 с.
4. Голуб Дж., Ван Лоун Ч. Матричные вычисления. Пер. с англ. / Под ред. Воеводина В.В. –М.: Мир, 1999. - 548 с.

Листинг программы

```
#include <iostream>
#include <string>
#include <ctime>
#include <fstream>
#include <cmath>
#include <algorithm>
#include <chrono>
using namespace std;

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "ru");

    auto startTime = chrono::high_resolution_clock::now();
    //Формула gaxpy z = A*x + y
    //Заем размерность матрицы
    const int n = 10000;
    const int m = n;

    cout << "Matrix shape: n=" << n << " and m=" << m << "\n";
    cout << "---------\n";
    //Созаем матрицу с заданной размерностью
    auto* matrix = new float[n * m];


    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            float r = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
            matrix[i * m + j] = floor(r * 10000000) / 10000000;
            //cout << matrix[i * m + j] << " ";
        }
        cout << "\n";
    }
    cout << "---------\n";
    cout << "\n";


    // Вектор
    cout << "Вектор x: \n";
    cout << "---------\n";
    float vector[m];

    for (int i = 0; i < m; i++)
    {
        float r = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
        vector[i] = floor(r * 10000000) / 10000000;
        //cout << vector[i] << "\n";
    }
    cout << "---------\n";
    cout << "\n";


    // второй вектор
    cout << "Вектор y: \n";
    cout << "---------\n";
    float vectorNext[n];

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

```
    {
        float r = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
        vectorNext[i] = floor(r * 10000000) / 10000000;
        //cout << vectorNext[i] << "\n";
    }
    cout << "---------\n";
    cout << "\n";



    //Умножение матрицы на вектор
    float outputMatrixV[n]{};

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            outputMatrixV[i] += matrix[i * m + j] * vector[j];
        }
    }
    //Прибавляем вектор, получаем итоговый вектор, пишем в фаил
    cout << "Выходной итоговый вектор после gaxpy: \n";
    cout << "---------\n";
    float output[n];

    for (int i = 0; i < n; i++)
    {
        output[i] = outputMatrixV[i] + vectorNext[i];
        cout << output[i] << "\n";
    }
    cout << "---------\n";

    auto endTime = chrono::high_resolution_clock::now();
    chrono::duration<float> duration = endTime - startTime;
    cout << "Duration " << duration.count() <<"s" << endl;



    string mytime = "Duration " + to_string(duration.count()) + "s " + "Matrix:" +
to_string(n) + "x" + to_string(m) + "\n";
    ofstream streamText;
    streamText.open("C:\\Users\\Dark_Monk\\Desktop\\gaxpy\\Time.txt", ofstream::app);
    //std::replace(mytime.begin(), mytime.end(), ',', '.'); // replace all ',' to '.'
    streamText << mytime;

    delete[] matrix;
}
```