МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Векторизация вычислений»

студента 2 курса, группы 20205

Муратова Максима Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Доцент
Власенко А. Ю.

Новосибирск 2021

СОДЕРЖАНИЕ

ЦЕЛЬ.....3
ЗАДАНИЕ.....3
ОПИСАНИЕ РАБОТЫ.....4
ЗАКЛЮЧЕНИЕ.....6
Приложение 1. `main.cpp` – *основной исполняемый файл*.....7
Приложение 2. `mtrx.h` – *заголовочный файл класса Matrix*.....9
Приложение 3. `mtrx.cpp` – *реализация основных методов класса Matrix*.....10
Приложение 4. `direct.cpp` – *реализация сложения и умножения “в лоб”*.....12
Приложение 5. `imm.h` – *реализация сложения и умножения через интринсики SSE*.....14
Приложение 6. `blas.h` – *реализация сложения и умножения через библиотеку BLAS*.....16
Приложение 7. `gen.sh` – *скрипт генерации исполняемых файлов*.....18
Приложение 8. `testing.sh` – *скрипт замера времени выполнения разных версий программы*.....19
Приложение 9. *Полезные команды*.....20

ЦЕЛЬ

- *На примере конкретного алгоритма провести векторизацию – вручную и при помощи специальной библиотеки – и сравнить его время выполнения до и после векторизации.*

ЗАДАНИЕ

1. *Написать различные версии программы следующего алгоритма:*

   Алгоритм обращения матрицы A размером $N \times N$ с помощью разложения в ряд: $A^{-1} = (I + R + R^2 + \ldots)B$, где $R = I - BA$, $B = \frac{A^T}{\|A\|_1 \cdot \|A\|_\infty}$, $\|A\|_1 = \max_j \sum_i |A_{ij}|$, $\|A\|_\infty = \max_i \sum_j |A_{ij}|$, I – единичная матрица (на главной диагонали – единицы, остальные – нули). Параметры алгоритма: N – размер матрицы, M – число членов ряда (число итераций цикла в реализации алгоритма).

   Программа должна быть в 3 версиях: без векторизации, через интинсики Intel и при помощи библиотеки BLAS

2. *Сравнить времена выполнения данных программ между собой*

ОПИСАНИЕ РАБОТЫ

Для удобства составления программы она была поделена на 6 частей:

1. `main.cpp` – собственно реализация алгоритма обращения матрицы (ПРИЛОЖЕНИЕ 1)
2. `mtrx.h, mtrx.cpp` – класс `Matrix`, который реализует интерфейс матрицы, кроме сложения, вычитания, умножения на скаляр и другую матрицу (ПРИЛОЖЕНИЯ 2, 3)
3. `direct.cpp, imm.cpp, blas.cpp` – реализация методов класса `Matrix` сложения, вычитания, умножений «в лоб», через интринсики и через библиотеку BLAS (ПРИЛОЖЕНИЯ 4, 5, 6 соответственно)

Коротко об использованных методах

1. Напрямую. Тривиальная реализация матричного умножения $c_{ij}=\sum_{k=1}^{n} a_{ik} b_{kj}$, *где* n – *порядок матриц* через три цикла, что даёт сложность алгоритма O(n^3), что означает, что время выполнения алгоритма будет стремительно возрастать. О том, сколько времеми ушло на данное решение, будет сказано чуть ниже.
2. Интринсики. Были использованы SSE интринсики, при помощи которых можно одновременно обрабатывать по 4 пары `float` чисел. Изначально планировалось использовать AVX интринсики, которые позволили бы обрабатывать 8 пар одновременно, но ни моё устройство, ни сервер их не поддерживали.
3. BLAS (Basic Linear Algebra Subprograms – базовые подпрограммы линейной алгебры). Данная библиотека была использована с целью противопоставления ручной векторизации, к тому же библиотека неплохо оптимизирована. Насколько – будет сказано ниже.

Частично код был написан на моём устройстве, чтобы можно было его проверить «локально», чтобы уже на сервере не пришлось бы исправлять ошибки, перегонять код из файла в файл или бороться с утечками памяти. Для обмена кодом использовалась платформа GitHub, через репозиторий https://github.com/mu2so4/evmpu-lab5
Команды для обмена файлами, компиляции файлов и тестирования программ даны в ПРИЛОЖЕНИИ 9.

Для тестов использовались 3 матрицы порядка 2048, с длиной ряда 10. Элементы матриц сгенерированы при помощи генератора случайных чисел с диапазоном от -100 до 100, с шагом 0,1 . Приводить сами тесты в настоящем отчёте не имеет смысла, поскольку они заняли бы слишком много места

Для собственно теста использовался скрипт `testing.sh`, написанный на bash. С его содержимым можно ознакомиться в ПРИЛОЖЕНИИ 8.

В результате я получил следующие данные о времени в секундах, необходимом для выполнения разных версий программы на сервере

1. Прямая реализация: **3013,99**
2. Через интринсики: **209,69**
3. Через BLAS: **11,93**

Изначально планировалось протестировать прямую реализацию на всех трёх матрицах, но через один прогон длиной 50 минут пришлось отказаться. Для остальных программ был выбран лучший из трёх результатов. Замерялось только user time, то есть время, которое ушло только на данную программу, без учёта других программ

Ещё было желание протестировать программы на своём компьютере, но одинарное матричное умножение «в лоб» (4 минуты на сервере vs 22 минуты на моём компьютере) показало, что выполнение программы в лоб заняло бы целую вечность – 4,5 часа. И от этого тоже пришлось отказаться.

Теперь сравним данные:

- Интринсики SSE в 14,4 раза быстрее «лобового» решения
- BLAS в 254 раза быстрее «лобового» решения
- BLAS в 17,6 раза быстрее интринсиков SSE

Напомню, что это актуально для данного алгоритма – алгоритма обращения матрицы через ряды.

ЗАКЛЮЧЕНИЕ

1. *Обращение матрицы «в лоб» на моём компьютере могло бы занять очень много времени, за которое можно было бы съездить из Академгородка в город и обратно (возможно, даже дважды)*
2. *Ручная векторизация (моей реализации), конечно, может помочь существенно ускорить выполнение данного алгоритма, но не настолько быстро, насколько это способна библиотека BLAS*
3. *Возможно, предыдущие два пункта вызваны моими кривыми руками или тем, что железо очень старое (2018 года выпуска)*

Приложение 1. *main.cpp*

```
#include <fstream>
#include "mtrx.h"

int main(int argc, char **argv) {
    if(argc != 3) {
        std::cerr << "Wrong count of parameters\n";
        return 0;
    }
    std::ifstream in(argv[1]);
    std::ofstream out(argv[2]);

    int size, sequencesCount;
    in >> size >> sequencesCount;
    auto *a = new Matrix{size};

    a->init(in);
    auto *b = new Matrix{*a};
    std::pair<float, float> maxes{a->getMaxes()};
    b->transport();
    *b *= 1.f / maxes.first / maxes.second;
    Matrix r{size}, res{size};
    Matrix b0{*b};
    *b *= *a;
    delete a;
    r -= *b;
    delete b;
    Matrix r0{r};
    res += r;
```

```
    for(int index = 0; index < sequencesCount - 1; index++) {
        r *= r0;
        res += r;
    }

    res *= b0;
    res.printData(out);
    in.close();
    out.close();
    return 0;
}
```

Приложение 2. `mtrx.h`

```
#ifndef VECTOR_MTRX_H
#define VECTOR_MTRX_H

#include <iostream>

class Matrix {
    int size;
    float *data;
public:
    explicit Matrix(int length);
    ~Matrix();
    Matrix(const Matrix &otherMatrix);

    void init(std::istream &input);
    void printData(std::ostream &output) const;
    void transport();
    std::pair<float, float> getMaxes() const;

    Matrix &operator+=(const Matrix &b);
    Matrix &operator-=(const Matrix &b);
    Matrix &operator*=(const Matrix &b);
    Matrix &operator*=(float mul);
};
#endif //VECTOR_MTRX_H
```

Приложение 3. *mtrx.cpp*

```
#include "mtrx.h"

Matrix::Matrix(int length): size(length) {
    data = new float[size * size];
    for(int index = 0; index < size; index++) {
        data[index * (size + 1)] = 1;
    }
}

Matrix::~Matrix() {
    delete[] data;
    data = nullptr;
}

Matrix::Matrix(const Matrix &otherMatrix):
        size(otherMatrix.size)
{
    data = new float[size * size];
    for(int index = 0; index < size * size; index++)
        data[index] = otherMatrix.data[index];
}

void Matrix::init(std::istream &input) {
    for(int index = 0; index < size * size; index++)
        input >> data[index];
}

void Matrix::printData(std::ostream &output) const {
    output << std::fixed;
    output.precision(2);
    for(int i = 0; i < size; i++) {
```

```
        for(int j = 0; j < size; j++) {
            output << data[size * i + j] << "\t";
        }
        output << '\n';
    }
}

void Matrix::transport() {
    for(int i = 0; i < size; i++) {
        for(int j = i + 1; j < size; j++) {
            std::swap(data[size * i + j],
                    data[size * j + i]);
        }
    }
}

std::pair<float, float> Matrix::getMaxes() const {
    float maxRow = .0, maxColumn = .0;
    for(int i = 0; i < size; i++) {
        float rows = .0, columns = .0;
        for(int j = 0; j < size; j++) {
            rows += data[i * size + j] >= 0 ?
                data[i * size + j] : -data[i * size + j];
            columns += data[j * size + i] >= 0 ?
                data[j * size + i] : -data[j * size + i];
        }
        if(rows > maxRow) maxRow = rows;
        if(columns > maxColumn) maxColumn = columns;
    }
    return std::pair<float, float>{maxRow, maxColumn};
}
```

Приложение 4. *direct.cpp*

```
#include "mtrx.h"

Matrix &Matrix::operator+=(const Matrix &b) {
    if(b.size == size)
        for(int index = 0; index < size * size; index++)
            data[index] += b.data[index];
    return *this;
}

Matrix &Matrix::operator-=(const Matrix &b) {
    if(b.size == size)
        for(int index = 0; index < size * size; index++)
            data[index] -= b.data[index];
    return *this;
}

Matrix &Matrix::operator*=(const Matrix &b) {
    if(size == b.size) {
        auto *newData = new float[size * size];
        if(size == 4) newData[4] = 0; //else I will get a nan
        for(int x = 0; x < size; x++) {
            for(int y = 0; y < size; y++) {
                for(int i = 0; i < size; i++)
                    newData[x * size + y] +=
                            data[x * size + i] *
                            b.data[i * size + y];
            }
        }
        delete[] data;
        data = newData;
    }
```

```
    return *this;
}

Matrix &Matrix::operator*=(float mul) {
    for(int index = 0; index < size * size; index++)
        data[index] *= mul;
    return *this;
}
```

Приложение 5. *imm.cpp*

```
#include <immintrin.h>
#include "mtrx.h"

Matrix &Matrix::operator+=(const Matrix &b) {
    if(size == b.size) {
        for(int index = 0; index + 4 <= size * size;
                index += 4) {
            _mm_store_ps(data + index,
                    _mm_add_ps(_mm_load_ps(data + index),
                    _mm_load_ps(b.data + index)));
        }
    }
    return *this;
}

Matrix &Matrix::operator-=(const Matrix &b) {
    if(size == b.size) {
        for(int index = 0; index + 4 <= size * size;
                index += 4) {
            _mm_store_ps(data + index,
                    _mm_sub_ps(_mm_load_ps(data + index),
                    _mm_load_ps(b.data + index)));
        }
    }
    return *this;
}

Matrix &Matrix::operator*=(const Matrix &b) {
    if(size == b.size) {
        auto *newData = new float[size * size];
        if(size == 4) newData[4] = 0; //else I will get a nan
```

```
        for(int row = 0; row < size; row++) {
            for(int column = 0; column < size; column++) {
                __m128 muller = _mm_set1_ps(
                        data[size * row + column]);
                for(int position = 0; position + 4 <= size;
                        position += 4) {
                    _mm_store_ps(newData + size * row +
                            position, _mm_add_ps
                            (_mm_load_ps(newData + size *
                            row + position),
                            _mm_mul_ps(_mm_load_ps(b.data +
                            size * column + position),
                            muller)));
                }
            }
        }

        delete[] data;
        data = newData;
    }
    return *this;
}

Matrix &Matrix::operator*=(float mul) {
    __m128 muls = _mm_set1_ps(mul);
    for(int index = 0; index + 4 <= size * size; index += 4)
{
        _mm_store_ps(data + index,
             _mm_mul_ps(_mm_load_ps(data + index), muls));
    }
    return *this;
}
```

Приложение 6. *blas.cpp*

```
#include "mtrx.h"
#include <cblas.h>

Matrix &Matrix::operator+=(const Matrix &b) {
    //матрицы как вектора: x = x + lambda * y,
    //сложение для каждого элемента
    cblas_saxpy(size * size, 1., b.data, 1, data, 1);
}

Matrix &Matrix::operator-=(const Matrix &b){
    //матрицы как вектора: x = x + lambda * y,
    //сложение для каждого элемента
    cblas_saxpy(size * size, -1., b.data, 1, data, 1);
}

Matrix &Matrix::operator*=(const Matrix &b){
    if(size == b.size) {
        auto *newData = new float[size * size];
        //поясню: строки главнее; обе матрицы не
        //транспортировать; число строк в data и newData;
        //число столбцов в b.data и newData; число столбцов
        //в data и строк в b.data; коэффициент умножения
        //произведения; ссылка на data; число элементов в
        //ведущей размерности в data (=число строк); ссылка
        //на b.data; число строк в b.data; коэффициент при
        //свободной матрице newData; ссылка на newData; число
        //строк в newData
        cblas_sgemm(CblasRowMajor, CblasNoTrans,
                CblasNoTrans, size, size, size, 1.0, data,
                size, b.data, size, 0.0, newData, size);
        delete[] data;
```

```
        data = newData;
    }
    return *this;
}

Matrix &Matrix::operator*=(float mul) {
    //матрица как вектор: размерность size*size,
    //множитель – mul, с шагом 1 (то есть умножаем всё)
    cblas_sscal(size * size, mul, data, 1);
}
```

Приложение 7. *gen.sh*

```
#!/bin/bash
g++ -std=c++11 main.cpp mtrx.h mtrx.cpp direct.cpp
    -o prog-direct
g++ -std=c++11 -msse main.cpp mtrx.h mtrx.cpp imm.cpp
    -o prog-imm
g++ -std=c++11 main.cpp mtrx.h mtrx.cpp blas.cpp -I
    $HOME/Linux_P4SSE2/include -L $HOME/Linux_P4SSE2/lib
    -lcblas -latlas -o prog-blas
```

Приложение 8. *tester.sh*

```
#!/bin/bash

echo "" >> report.txt


echo "Direct test" >> report.txt

echo "Direct test"

\time -o report.txt -a -f "test1.txt: %U\n"
    ./prog-direct test1.txt output.txt

echo "test1 done"


echo "Imm test" >> report.txt

echo "Imm test"

\time -o report.txt -a -f "test1.txt: %U\n"
    ./prog-imm test1.txt output.txt

echo "test1 done"

\time -o report.txt -a -f "test2.txt: %U\n"
    ./prog-imm test2.txt output.txt

echo "test2 done"

\time -o report.txt -a -f "test3.txt: %U\n"
    ./prog-imm test3.txt output.txt

echo "test3 done"


echo "Blas test" >> report.txt

echo "Blas test"

\time -o report.txt -a -f "test1.txt: %U\n"
    ./prog-blas test1.txt output.txt

echo "test1 done"

\time -o report.txt -a -f "test2.txt: %U\n"
    ./prog-blas test2.txt output.txt

echo "test2 done"

\time -o report.txt -a -f "test3.txt: %U\n"
    ./prog-blas test3.txt output.txt

echo "test3 done"

echo "Testing successfully passed!"
```

- Копирование репозитория
  ```
  $ git clone https://github.com/mu2so4/evmpu-lab5
  ```
- Обновление данных в локальной директории
  ```
  $ cd 20205/Muratov/evmpu-lab5
  $ git pull origin master
  ```
- Фиксация изменений файла
  ```
  $ git add <filename>
  ```
- Коммит
  ```
  $ git commit -m "Some commit message"
  ```
- Загрузка изменений в облачную репозиторию (нужен токен)
  ```
  $ git push
  ```
- Сборка исполняемых файлов
  ```
  $ ./gen.sh
  ```
- Замер времени работы программ с записью результатов в `report.txt`
  ```
  $ ./tester.sh
  ```