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 ЛАБОРАТОРНОЙ РАБОТЫ

«ВЕКТОРИЗАЦИЯ ВЫЧИСЛЕНИЙ»

студента Бородина Артёма Максимовича 2 курса, 19205 группы

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н, доцент
А.Ю. Власенко

Новосибирск 2020

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Изучение SIMD-расширений архитектуры x86/x86-64.

2. Изучение способов использования SIMD-расширений в программах на языке Си.

3. Получение навыков использования SIMD-расширений.

ЗАДАНИЕ

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

1. Написать три варианта программы, реализующей алгоритм из задания:

2. Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.

3. Каждый вариант программы оптимизировать по скорости, насколько это возможно.

4. Сравнить время работы трех вариантов программы для N=2048, M=10.

5. Составить отчет по лабораторной работе.

ОПИСАНИЕ РАБОТЫ

1. Было изучено понятие векторизации вычислений, различные виды SIMD-расширений и их способы использования.

2. Были написаны 3 программа для обращения матрицы размера N*N:

3. Была проведена проверка правильности работы каждой программы (Приложение 4). Погрешность получилась равной ~0.1%.

4. Были проведены замеры времени работы каждой программы с несколькими размерами матриц от N=128 до N=2048.(Приложение 5)

5. При сравнении становится заметно что программа с использованием библиотеки OpenBLAS справляется быстрее всего. Разница достигала почти 470 раз по отношению к времени программы с векторизацией и почти 1890 раз для программы без векторизации. Такая разница получилась из-за того, что значительная часть библиотеки OpenBLAS написана на ассемблере и оптимизирована под различные типы процессоров. Также эта библиотека наиболее эффективно использует кэш процессора.

6. Разница между программы с векторизацией и программы без примерно в 4 раза. Это происходить из-за того, что в первой программе операции производятся сразу над векторами размера 4, что позволяет ускорить операции примерно в 4 раза.

ЗАКЛЮЧЕНИЕ

Во время выполнения лабораторной работы я получил практические знания работы с векторизацией в коде программы и исследовал эффективность библиотеки OpenBLAS.

Приложение 1. *Программа без векторизации*

```
#include <ctime>
#include <iostream>

#define N 2048
#define PRINT_N 4

void PrintMat(float A[N][N]) {
    for (int i = 0; i < PRINT_N; ++i) {
        for (int j = 0; j < PRINT_N; ++j) {
            std::cout << A[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

void MultiplyMatrices(const float A[N][N], const float B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void SumMatrices(const float A[N][N], const float B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

void SubMatrices(const float A[N][N], const float B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}

void InverseMatrix(float A[N][N], float reversedMatrix[N][N]) {
    float maxColSum = 0, maxRowSum = 0;
    float I[N][N] = {0};
    float B[N][N] = {0};
    float R[N][N] = {0};
    float Rtmp[N][N] = {0};
    float tmpMatrix[N][N] = {0};
    float matrixSum[N][N] = {0};
    int i, j;

    for (i = 0; i < N; i++) { // Max sum per column and per row
        float tmpSum1 = 0;
        float tmpSum2 = 0;
        for (j = 0; j < N; j++) {
            tmpSum1 = tmpSum1 + A[i][j];
            tmpSum2 = tmpSum2 + A[j][i];
        }
        if (tmpSum1 > maxColSum) {
            maxColSum = tmpSum1;
        }
        if (tmpSum2 > maxRowSum) {
            maxRowSum = tmpSum2;
        }
    }
```

```
    for (i = 0; i < N; i++) { // Calculate B matrix
        for (j = 0; j < N; j++) {
            B[i][j] = A[j][i] / (maxColSum * maxRowSum);
        }
    }
    for (i = 0; i < N; i++) { // Create indent matrix
        I[i][i] = 1;
    }

    clock_t matMulTime = clock();
    MultiplyMatrices(B, A, tmpMatrix);// tmpMatrix = BA
    matMulTime = clock() - matMulTime;
    std::cout << "matMulTime: " << matMulTime / (float) CLOCKS_PER_SEC << std::endl;

    SubMatrices(I, tmpMatrix, R); // R = I - BA
    SubMatrices(I, tmpMatrix, Rtmp); // Rtmp = R

    SumMatrices(I, R, matrixSum);
    for (int n = 0; n < 10; n++) { // I + R + R^2 +...
        MultiplyMatrices(R, Rtmp, tmpMatrix); // R^(n+1)
        SumMatrices(tmpMatrix, matrixSum, matrixSum); // matrixSum += R^(n+1)
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                Rtmp[i][j] = tmpMatrix[i][j];
            }
        }
    }
    MultiplyMatrices(matrixSum, B, reversedMatrix); // A^(-1) = (I + R + R^2 + ...)B
}

int main() {
    float inputMatrix[N][N];
    float outputMatrix[N][N];
    srand(time(nullptr));
    for (auto &row : inputMatrix) { // Initialize some matrix
        for (float &element : row) {
            element = int(rand()) % 10;
        }
    }
    /*for (int i = 0; i < N; i++) { // Initialize test matrix
        for (int j = 0; j < N; j++) {
            if (i == j) {
                inputMatrix[i][j] = 10;
            } else {
                inputMatrix[i][j] = 1;
            }
        }
    }*/

    PrintMat(inputMatrix);
    clock_t elapsedTime = clock();
    InverseMatrix(inputMatrix, outputMatrix);
    elapsedTime = clock() - elapsedTime;
    PrintMat(outputMatrix);

    std::cout << "All time: " << elapsedTime / (float) CLOCKS_PER_SEC << std::endl;
    return 0;
}
```

Приложение 2. *Программа с векторизацией из xmmintrin.h*

```
#include <ctime>
#include <xmmintrin.h>
#include <iostream>
#include <algorithm>

#define N 2048
#define PRINT_N 4

void PrintMat(float A[N][N]) {
    for (int i = 0; i < PRINT_N; ++i) {
        for (int j = 0; j < PRINT_N; ++j) {
            std::cout << A[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

void TransposeMatrix(float A[N][N]) {
    for (int i = 0; i < N; i += 4) {
        for (int j = 0; j < N; j += 4) {
            if (i >= j) {
                __m128 row11 = _mm_loadu_ps(&A[i + 0][j]);
                __m128 row12 = _mm_loadu_ps(&A[i + 1][j]);
                __m128 row13 = _mm_loadu_ps(&A[i + 2][j]);
                __m128 row14 = _mm_loadu_ps(&A[i + 3][j]);
                _MM_TRANSPOSE4_PS(row11, row12, row13, row14);
                __m128 row21 = _mm_loadu_ps(&A[j + 0][i]);
                __m128 row22 = _mm_loadu_ps(&A[j + 1][i]);
                __m128 row23 = _mm_loadu_ps(&A[j + 2][i]);
                __m128 row24 = _mm_loadu_ps(&A[j + 3][i]);
                _MM_TRANSPOSE4_PS(row21, row22, row23, row24);
                _mm_storeu_ps(&A[i + 0][j], row21);
                _mm_storeu_ps(&A[i + 1][j], row22);
                _mm_storeu_ps(&A[i + 2][j], row23);
                _mm_storeu_ps(&A[i + 3][j], row24);
                _mm_storeu_ps(&A[j + 0][i], row11);
                _mm_storeu_ps(&A[j + 1][i], row12);
                _mm_storeu_ps(&A[j + 2][i], row13);
                _mm_storeu_ps(&A[j + 3][i], row14);
            }
        }
    }
}

void MultiplyMatrices(float A[N][N], float B[N][N], float C[N][N]) {
    TransposeMatrix(B);
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; j += 4) {
            __m128 vR = _mm_setzero_ps();
            for (int k = 0; k < N; k++) {
                __m128 vA = _mm_set1_ps(A[i][k]);
                __m128 vB = _mm_loadu_ps(&B[k][j]);
                vR = _mm_add_ps(vR, _mm_mul_ps(vA, vB));
            }
            _mm_storeu_ps(&C[i][j], vR);
        }
    }
    TransposeMatrix(B);
}

void SumMatrices(float A[N][N], float B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int k = 0; k < N; k += 4) {
            __m128 vA = _mm_loadu_ps(&A[i][k]);
            __m128 vB = _mm_loadu_ps(&B[i][k]);
            _mm_storeu_ps(&C[i][k], _mm_add_ps(vA, vB));
        }
    }
}

void SubMatrices(float A[N][N], float B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int k = 0; k < N; k += 4) {
            __m128 vA = _mm_loadu_ps(&A[i][k]);
            __m128 vB = _mm_loadu_ps(&B[i][k]);
            _mm_storeu_ps(&C[i][k], _mm_sub_ps(vA, vB));
        }
    }}
```

```
float VectorFindMax(__m128 vA) {
    float result[4];
    _mm_storeu_ps(result, vA);
    return *std::max_element(result, result + 4);
}

float FindMaxColSum(float A[N][N]) {
    float max = 0;
    for (int i = 0; i < N; i += 4) {
        __m128 row1;
        __m128 vA = _mm_setzero_ps();
        for (int j = 0; j < N; j++) {
            row1 = _mm_loadu_ps(&A[j][i]);
            vA = _mm_add_ps(vA, row1);
        }
        if (VectorFindMax(vA) > max) {
            max = VectorFindMax(vA);
        }
    }
    return max;
}

void InvertMatrix(float A[N][N], float reversedMatrix[N][N]) {
    float I[N][N] = {0};
    float B[N][N] = {0};
    float R[N][N] = {0};
    float Rtmp[N][N] = {0};
    float tmpMatrix[N][N] = {0};
    float matrixSum[N][N] = {0};
    int i, j;

    float maxColSum = FindMaxColSum(A);
    TransposeMatrix(A);
    float maxRowSum = FindMaxColSum(A);
    for (i = 0; i < N; i++) { // Calculate B matrix
        for (j = 0; j < N; j += 4) {
            __m128 vA = _mm_loadu_ps(&A[i][j]);
            __m128 vB = _mm_set_ps1(maxColSum * maxRowSum);
            _mm_storeu_ps(&B[i][j], _mm_div_ps(vA, vB));
        }
    }
    TransposeMatrix(A);
    for (i = 0; i < N; i++) { // Create indent matrix
        I[i][i] = 1;
    }

    clock_t matMulTime = clock();
    MultiplyMatrices(B, A, tmpMatrix);// tmpMatrix = BA
    matMulTime = clock() - matMulTime;
    std::cout << "matMulTime: " << matMulTime / (float) CLOCKS_PER_SEC << std::endl;

    SubMatrices(I, tmpMatrix, R); // R = I - BA
    SubMatrices(I, tmpMatrix, Rtmp); // Rtmp = R

    SumMatrices(I, R, matrixSum);
    for (int n = 0; n < 10; n++) { // I + R + R^2 +...
        MultiplyMatrices(R, Rtmp, tmpMatrix); // R^(n+2)
        SumMatrices(tmpMatrix, matrixSum, matrixSum); // matrixSum += R^(n+2)
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j += 4) {
                __m128 vA = _mm_loadu_ps(&tmpMatrix[i][j]);
                _mm_storeu_ps(&Rtmp[i][j], vA);
            }
        }
    }
    MultiplyMatrices(matrixSum, B, reversedMatrix); // A^(-1) = (I + R + R^2 + ...)B
}
```

```cpp
int main() {
    float inputMatrix[N][N];
    float outputMatrix[N][N];
    srand(time(nullptr));
    for (auto &row : inputMatrix) { // Initialize some matrix
        for (float &element : row) {
            element = int(rand()) % 10;
        }
    }
    /*for (int i = 0; i < N; i++) { // Initialize test matrix
        for (int j = 0; j < N; j++) {
            if (i == j) {
                inputMatrix[i][j] = 10;
            } else {
                inputMatrix[i][j] = 1;
            }
        }
    }*/

    PrintMat(inputMatrix);
    clock_t elapsedTime = clock();
    InvertMatrix(inputMatrix, outputMatrix);
    elapsedTime = clock() - elapsedTime;
    PrintMat(outputMatrix);

    std::cout << "All time: " << elapsedTime / (float) CLOCKS_PER_SEC << std::endl;
    return 0;
}
```

Приложение 3. *Программа с библиотекой OpenBLAS*

```
#include <ctime>
#include <iostream>
#include "cblas.h"

#define N 2048
#define PRINT_N 4

void PrintMat(float A[N * N]) {
    for (int i = 0; i < PRINT_N; ++i) {
        for (int j = 0; j < PRINT_N; ++j) {
            std::cout << A[i * N + j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

void InverseMatrix(float A[N * N], float reversedMatrix[N * N]) {
    float maxColSum = 0, maxRowSum = 0;
    float B[N * N] = {0};
    float R[N * N] = {0};
    float Rtmp[N * N] = {0};
    float Rmul[N * N] = {0};
    float matrixSum[N * N] = {0};
    int i;

    float vector[N] = {0};
    float colSum[N] = {0};

    std::fill(vector, vector + N, 1);
    cblas_sgemv(CblasRowMajor, CblasNoTrans, N, N,
                1, A, N, vector, 1, 0, colSum, 1); // colSum = A * vector
    maxColSum = colSum[cblas_isamax(N, colSum, 1)];

    std::fill(vector, vector + N, 1);
    cblas_sgemv(CblasRowMajor, CblasTrans, N, N,
                1, A, N, vector, 1, 0, colSum, 1); // rowSum = A(-1) * vector
    maxRowSum = colSum[cblas_isamax(N, colSum, 1)];

    cblas_scopy(N * N, A, 1, B, 1);
    cblas_sscal(N * N, 1 / (maxColSum * maxRowSum), B, 1); // Calculate B matrix

    for (i = 0; i < N; i++) { // Create indent matrices
        matrixSum[i * N + i] = 1;
        Rtmp[i * N + i] = 1;
    }

    clock_t matMulTime = clock();
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N,
                -1, B, N, A, N, 1, Rtmp, N); // Rtmp = -BA + Rtmp
    matMulTime = clock() - matMulTime;
    std::cout << "matMulTime: " << matMulTime / (float) CLOCKS_PER_SEC << std::endl;
    cblas_scopy(N * N, Rtmp, 1, R, 1); // R = Rtmp

    for (i = 0; i < 11; i++) { // I + R + R^2 + ..
        cblas_saxpy(N * N, 1, Rtmp, 1, matrixSum, 1); // matrixSum += Rtmp
        cblas_scopy(N * N, Rtmp, 1, Rmul, 1); // Rmul = Rtmp
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N,
                    1, Rmul, N, R, N, 0, Rtmp, N); // Rtmp = R^(i+1)
    }
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N,
                1, matrixSum, N, B, N, 0, reversedMatrix, N); // A^(-1) = (I + R + R^2 + ...)B
}
```

```
int main() {
    float inputMatrix[N * N];
    float outputMatrix[N * N];
    srand(time(nullptr));
    for (int i = 0; i < N; i++) { // Initialize some matrix
        for (int j = 0; j < N; j++) {
            inputMatrix[i * N + j] = int(rand()) % 10;
        }
    }
    /*for (int i = 0; i < N; i++) { // Initialize test matrix
        for (int j = 0; j < N; j++) {
            if (i == j) {
                inputMatrix[i * N + j] = 10;
            } else {
                inputMatrix[i * N + j] = 1;
            }
        }
    }*/

    PrintMat(inputMatrix);
    clock_t elapsedTime = clock();
    InverseMatrix(inputMatrix, outputMatrix);
    elapsedTime = clock() - elapsedTime;
    PrintMat(outputMatrix);

    std::cout << "All time: " << elapsedTime / (float) CLOCKS_PER_SEC << std::endl;
    return 0;
}
```

Приложение 4. *Проверка корректности программ*

Матрица
10 1 1 1
1 10 1 1
1 1 10 1
1 1 1 10
Библиотека OpenBLAS
0.102531 -0.00853598 -0.00853597 -0.00853597
-0.00853598 0.102531 -0.00853597 -0.00853597
-0.00853597 -0.00853597 0.102531 -0.00853597
-0.00853597 -0.00853597 -0.00853597 0.102531
Векторизация из *xmmintrin.h*
0.102531 -0.00853597 -0.00853597 -0.00853597
-0.00853597 0.102531 -0.00853597 -0.00853597
-0.00853598 -0.00853598 0.102531 -0.00853597
-0.00853597 -0.00853597 -0.00853597 0.102531
Без векторизации
0.102531 -0.00853598 -0.00853597 -0.00853597
-0.00853598 0.102531 -0.00853597 -0.00853597
-0.00853597 -0.00853597 0.102531 -0.00853597
-0.00853597 -0.00853597 -0.00853597 0.102531
Точный расчет
0.1025641025 -0.0085470085 -0.0085470085 -0.0085470085
-0.0085470085 0.1025641025 -0.0085470085 -0.0085470085
-0.0085470085 -0.0085470085 0.1025641025 -0.0085470085
-0.0085470085 -0.0085470085 -0.0085470085 0.1025641025

Приложение 5. *Замеры времени*

	128	256	512	1024	2048
БЕЗ ВЕКТОРИЗАЦИИ	0,149	1,394	13,04	631,042	4849,85
С ВЕКТОРИЗАЦИЕЙ	0,086	0,574	4,707	157,849	1199,94
OPENBLAS	0,006	0,013	0,087	0,407	2,568