**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 21204

**Осипова Александра Александровича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
Власенко А.Ю.

Новосибирск 2022

# 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

1. Написать три варианта программы, реализующей алгоритм обращения матрицы A размером N x N с помощью разложения в ряд

$$A^{-1} = (I + R + R^2 + \ldots)B, \quad \text{где} \quad R = I - BA, \quad B = \frac{A^T}{\|A\|_1 \cdot \|A\|_\infty}, \quad \|A\|_1 = \max_j \sum_i |A_{ij}|, \quad \|A\|_\infty = \max_i \sum_j |A_{ij}|,$$

   I – единичная матрица, N – размер матрицы, M – число членов ряда.

   Варинты программы:

   - вариант без векторизации
   - вариант с ручной векторизацией
   - вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS.

2. Проверить правильность работы программы на нескольких небольших тестовых наборах входных данных.
3. Каждый вариант программы оптимизировать по скорости.
4. Сравнить время работы трех вариантов программы для N=2048, M=10.
5. Составить отчет по лабораторной работе.

# ОПИСАНИЕ РАБОТЫ

Для чистоты эксперименты все программы будут запускаться на сервере кафедры.

Перед реализацией самого алгоритма была написана функция `void SetRandValues(int N)`, записывающая в текстовый файл 4 194 304 (2048 * 2048) вещественных числа, из которых потом будет состоять матрица A.

```
void SetRandValues(int N) {
    std::ofstream input;
    input.open("matrix.txt");
    if (input.is_open()) {
        for (int i = 0; i < N; ++i) {
            float value = (float)rand() / float(RAND_MAX % 1000);
            if (rand() % 2 == 0) {
                value *= -1.0;
            }
            input << std::round(value * 100)/100 << std::endl;
        }
    }
}
```

1. Вариант без векторизации.

Для всех трех вариантов был написан class Matrix (листинг программы см. Приложение 1), реализующий работу с матрицей. Данный класс хранит float* matrix (для построчного хранения матрицы в виде массива) и size_t N (размер матрицы). Класс реализует следующие методы:

```
Matrix(int N); //конструктор, который строит нулевую матрицу размера N
void init(std::ifstream& input); //заполняет матрицу числами из файла
void output(); //печатает матрицу
int getN(); //геттер для поля size_t N
float getCell(int i, int j); //возвращает значение ячейки A[i][j]
void setCell(int i, int j, float value); //устанавливает значение в ячейку A[i][j]
Matrix transpose();   //транспонирует матрицу
Matrix invert(int M); //обращает матрицу
```

Так же были добавлены перегруженные арифметические операторы для сложения, вычитания и умножения матриц.

Реализация перегруженного оператора сложения для двух матриц:

```
Matrix operator+(const Matrix& left, const Matrix& right) {
        Matrix result(left.N);
        for (int i = 0; i < left.N * left.N; ++i) {
              result.matrix[i] = left.matrix[i] + right.matrix[i];
        }
        return result;
}
```

Реализация перегруженного оператора вычитания для двух матриц:

```
Matrix operator-(const Matrix& left, const Matrix& right) {
        int N = left.N;
```

```
        Matrix result(N);
        for (int i = 0; i < N * N; ++i) {
              result.matrix[i] = left.matrix[i] - right.matrix[i];
        }
        return result;
}
```

Реализация метода, транспонирующего матрицу:

```
Matrix Matrix::transpose() {
        Matrix T(N);
        for (int i = 0; i < N; ++i) {
              for (int j = 0; j < N; ++j) {
                    T.matrix[i * N + j] = this->matrix[j * N + i];
              }
        }
        return T;
}
```

Реализация перегруженного оператора умножения для двух матриц:

```
Matrix operator*(const Matrix& left, const Matrix& right) {
        int N = left.N;
        Matrix result(N);
        for (int i = 0; i < N; ++i) {
              for (int k = 0; k < N; ++k) {
                    for (int j = 0; j < N; ++j) {
                          result.matrix[i * N + j] += left.matrix[i * N + k] *
  right.matrix[k * N + j];
                    }
              }
        }
        return result;
}
```

Функция, создающая вспомогательную матрицу B:

```
Matrix createMatrixB(Matrix& A) {
        Matrix B(A.getN());
        Matrix At = A.transpose();
        float A1 = getA1(A);
        float Ainf = getAinf(A);
        B = At / (A1 * Ainf);
        return B;
}
```

Реализация главного метода, обращающего исходную матрицу A:

```
Matrix Matrix::invert(int M) {
        Matrix B = createMatrixB(*this);
        Matrix I = createMatrixI(B.getN());
        Matrix R = I - B * (*this);
        Matrix powR = R;

        Matrix result = I;
        for (int i = 0; i < M; ++i) {
              result += R;
              R = powR * R;
        }

        result = result * B;
```

```
        return result;
}
```

Функция `int main()`. В ней поставлен таймер, определяющий, за какое время метод `Matrix Matrix::invert(int M)` обращает матрицу A. Для этих целей был выбран таймер системного времени `time(time_t* Time)`. Так же здесь выполняется проверка корректности обращения матрицы. Создается матрица E = A * I, где A – исходная матрица, I – обратная матрица к A. Если E – единичная матрица, то для нее нормы ||A1|| ~ ||Ainf|| ~ 1.

```
int main() {
        std::ifstream input;
        input.open("matrix.txt");
        if (input.is_open()) {
                Matrix A(SIZE);
                A.init(input);
                time_t start, end;
                time(&start);
                Matrix I = A.invert(10);
                time(&end);
                Matrix E = A * I;
                std::cout << "Time taken to find matrix I (inverted): " <<
  difftime(end, start) << "sec\n";
                std::cout << "E = A * I\n";
                std::cout << "|E_1| = " << getA1(E) << std::endl;
                std::cout << "|E_inf| = " << getAinf(E) << std::endl;
        }

        return 0;
}
```

Проверка корректности программы, при N=8, M=10000:

```
evmpu@comrade:~/21204/osipov/lab4$ ./default.out
Matrix A is initialized
Time taken to find matrix I (inverted): 0sec
E = A * I
0.999826 0.000239074 -8.72836e-05 2.2985e-05 -0.000162827 9.6947e-05 -5.43594e-05 2.83569e-05
0.000242501 0.999844 1.99974e-05 -5.11706e-05 0.000208694 -2.5928e-05 -1.30311e-05 -3.1434e-05
-8.82745e-05 1.96397e-05 0.999982 6.22422e-05 -2.05245e-05 -2.55704e-05 2.67327e-05 3.41237e-05
2.39611e-05 -4.80413e-05 5.89937e-05 0.999996 -1.41002e-05 -1.04904e-05 -1.57654e-05 -1.18315e-05
-0.000169635 0.000212133 -2.13832e-05 -1.10865e-05 0.999786 8.55923e-05 -4.01139e-05 -9.59635e-06
9.79602e-05 -2.59504e-05 -2.54847e-05 -9.80869e-06 8.43816e-05 1 -1.31577e-05 1.05798e-05
-5.35846e-05 -1.41263e-05 2.57343e-05 -1.70916e-05 -3.75053e-05 -1.33365e-05 1.00003 -2.43559e-05
3.02792e-05 -3.08156e-05 3.06368e-05 -1.30534e-05 -8.0131e-06 1.0252e-05 -2.42293e-05 0.999999

|E_1| = 1.00053
|E_inf| = 1.00052
```

Результат работы программы, запущенной на сервере кафедры, для N = 2048, M = 10:

```
evmpu@comrade:~/21204/osipov/lab4$ ./default.out
Matrix A is initialized
Time taken to find matrix I (inverted): 874sec
E = A * I
|E_1| = 0.851045
|E_inf| = 0.851044
evmpu@comrade:~/21204/osipov/lab4$
```

2. Вариант с ручной векторизацией.

Для этого варианта реализации программы были выбраны встроенные функции SIMD-расширений (SIMD intrinsic). Оптимизации подверглись перегруженные операторы сложения (см. Приложение 2), вычитания (см. Приложение 3) и умножения (см. Приложение 4). Для их реализации были использованы локальные переменные типа __m128 (а также указатели на них) для хранения четырех последовательных элементов матрицы, хранения проивездедений элементов матриц. Остальной код остался без изменений относительно варианта без векторизации.

Проверка корректности программы, при N=8, M=10000:

```
evmpu@comrade:~/21204/osipov/lab4$ ./intrinsic.out
Matrix A is initialized
Time taken to find matrix I (inverted): 1sec
E = A * I
0.999826 0.000239074 -8.72836e-05 2.2985e-05 -0.000162827 9.6947e-05 -5.43594e-05 2.83569e-05
0.000242501 0.999844 1.99974e-05 -5.11706e-05 0.000208694 -2.5928e-05 -1.30311e-05 -3.1434e-05
-8.82745e-05 1.96397e-05 0.999982 6.22422e-05 -2.05245e-05 -2.55704e-05 2.67327e-05 3.41237e-05
2.39611e-05 -4.80413e-05 5.89937e-05 0.999996 -1.41002e-05 -1.04904e-05 -1.57654e-05 -1.18315e-05
-0.000169635 0.000212133 -2.13832e-05 -1.10865e-05 0.999786 8.55923e-05 -4.01139e-05 -9.59635e-06
9.79602e-05 -2.59504e-05 -2.54847e-05 -9.80869e-06 8.43816e-05 1 -1.31577e-05 1.05798e-05
-5.35846e-05 -1.41263e-05 2.57343e-05 -1.70916e-05 -3.75053e-05 -1.33365e-05 1.00003 -2.43559e-05
3.02792e-05 -3.08156e-05 3.06368e-05 -1.30534e-05 -8.0131e-06 1.0252e-05 -2.42293e-05 0.999999

|E_1| = 1.00053
|E_inf| = 1.00052
```

Результат работы программы, запущенной на сервере кафедры, для N=2048, M=10:

```
evmpu@comrade:~/21204/osipov/lab4$ ./intrinsic.out
Matrix A is initialized
Time taken to find matrix I (inverted): 272sec
E = A * I
|E_1| = 0.851045
|E_inf| = 0.851044
evmpu@comrade:~/21204/osipov/lab4$
```

3. Вариант с использованием библиотеки BLAS

В данном варианте оптимизации также подверглись перегруженные операторы сложения (см. Приложение 5), вычитания (см. Приложение 6) и умножения (см. Приложение 6). Были использованы такие функции BLAS, как:

```
void cblas_scopy(const int __N, const float *__X, const int __incX, float *__Y,
const int __incY); //копирование матрицы X -> Y
```

```
void cblas_saxpy(const int __N, const float __alpha, const float *__X, const int
__incX, float *__Y, const int __incY); //сложение(вычитание) матриц

//Y[i] = (alpha * X[i]) + Y[i]
```

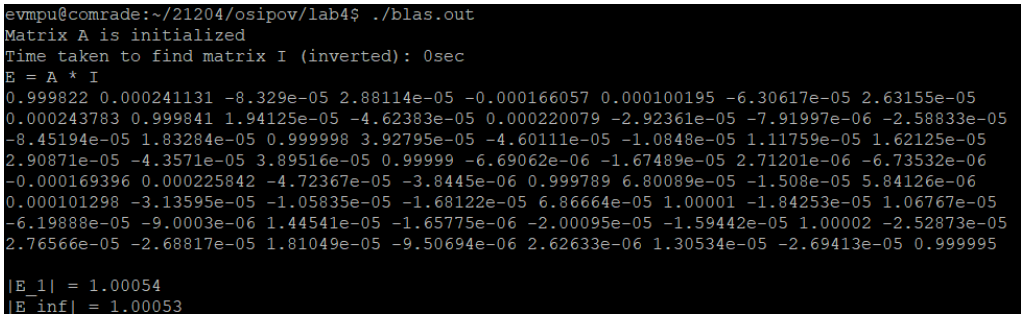
```
void cblas_sgemm(const enum CBLAS_ORDER __Order, const enum CBLAS_TRANSPOSE
__TransA, const enum CBLAS_TRANSPOSE __TransB, const int __M, const int __N, const
int __K, const float __alpha, const float *__A, const int __lda, const float *__B,
const int __ldb, const float __beta, float *__C, const int __ldc); //умножение

// C = αAB + βC
```

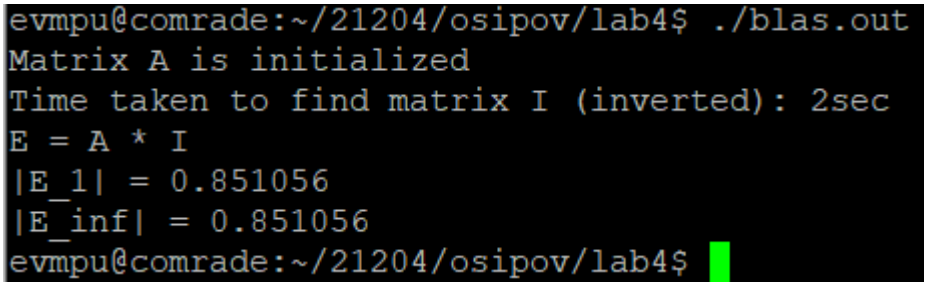
Команда для компиляции программы:

icpc blas.cpp -qmkl -diag-disable=10441 -o blas.out

Проверка корректности программы, при N=8, M=10000:




Результат работы программы, запущенной на сервере кафедры, для N=2048, M=10:



# ЗАКЛЮЧЕНИЕ

После написания трех программ были получены следующие результаты по времени:

| default.out | intrinsic.out | blas.out |
|---|---|---|
| 874 с | 272 с | 2 с |

Программа intrinsic.out показа результат в ~4 раза лучше, так как в ней используются 128 битные векторные регистры, каждый из которых может одновременно хранить 4 переменные типа float и выполнять над ними одну операцию, вместо четырех.

Программа blas.out оказалась на порядок быстрее, так как в ней использовались библиотечные функции BLAS.

# Приложение 1. Листинг программы без векторизации

```
#include <iostream>
#include <vector>
#include <fstream>
#include <cmath>
#include <time.h>
#define SIZE 2048

class Matrix {
public:
	Matrix();
	Matrix(int N);
	void init(std::ifstream& input);
	void output();
	int getN();
	float getCell(int i, int j);
	void setCell(int i, int j, float value);
	Matrix transpose();
	Matrix invert(int M);
	Matrix& operator=(const Matrix& right);
	Matrix& operator+=(const Matrix& right);
	Matrix& operator-=(const Matrix& right);
	friend Matrix operator*(const Matrix& left, const Matrix& right);
	friend Matrix operator+(const Matrix& left, const Matrix& right);
	friend Matrix operator-(const Matrix& left, const Matrix& right);
	friend Matrix operator/(const Matrix& left, float right);
private:
	float* matrix;
	size_t N;
};

Matrix::Matrix() : matrix(nullptr), N(0) {
}

Matrix::Matrix(int N) : N(N) {
	matrix = new float[N * N];

	for (int i = 0; i < N; ++i) {
		for (int j = 0; j < N; ++j) {
			matrix[i * N + j] = 0;
		}
	}
}

int Matrix::getN() {
	return N;
}

void Matrix::init(std::ifstream& input) {
	for (int i = 0; i < N; ++i) {
		for (int j = 0; j < N; ++j) {
			float value;
			input >> matrix[i * N + j];
		}
	}
	std::cout << "Matrix A is initialized\n";
}

void Matrix::output() {
	for (int i = 0; i < N; ++i) {
		for (int j = 0; j < N; ++j) {
			std::cout << matrix[i * N + j] << " ";
		}
```

```
            std::cout << "\n";
        }
        std::cout << "\n";
}

Matrix& Matrix::operator=(const Matrix& right) {

        if (this != &right){
                this->matrix = right.matrix;
                this->N = right.N;
         }
         return *this;
}

Matrix& Matrix::operator+=(const Matrix& right) {
        Matrix result = *this + right;
        *this = result;
        return *this;
}

Matrix& Matrix::operator-=(const Matrix& right) {
        for (int i = 0; i < N * N; ++i) {
                this->matrix[i] -= right.matrix[i];
        }
        return *this;
}

Matrix operator*(const Matrix& left, const Matrix& right) {
        int N = left.N;
        Matrix result(N);
        for (int k = 0; k < N; ++k) {
                for (int i = 0; i < N; ++i) {
                        for (int j = 0; j < N; ++j) {
                                result.matrix[i * N + j] += left.matrix[i * N + k] *
right.matrix[k * N + j];
                        }
                }
        }
        return result;
}

Matrix operator+(const Matrix& left, const Matrix& right) {
        Matrix result(left.N);
        for (int i = 0; i < left.N * left.N; ++i) {
                result.matrix[i] = left.matrix[i] + right.matrix[i];
        }
        return result;
}

Matrix operator-(const Matrix& left, const Matrix& right) {
        int N = left.N;
        Matrix result(N);
        for (int i = 0; i < N * N; ++i) {
                result.matrix[i] = left.matrix[i] - right.matrix[i];
        }
        return result;
}

Matrix operator/(const Matrix& left, float right) {
        Matrix result = left;
        for (int i = 0; i < left.N; ++i) {
                for (int j = 0; j < left.N; ++j) {
                        result.matrix[i * left.N + j] /= right;
                }
        }
```

```
	return result;
}

Matrix Matrix::transpose() {
	Matrix T(N);
	for (int i = 0; i < N; ++i) {
		for (int j = 0; j < N; ++j) {
			T.matrix[i * N + j] = this->matrix[j * N + i];
		}
	}
	return T;
}

float Matrix::getCell(int i, int j) {
	return this->matrix[i * N + j];
}

void Matrix::setCell(int i, int j, float value) {
	this->matrix[i * N + j] = value;
}

float getA1(Matrix& A) {
	Matrix trA = A.transpose();
	float A1 = 0;
	for (int i = 0; i < A.getN(); ++i) {
		float curMax = 0;
		for (int j = 0; j < A.getN(); ++j) {
			curMax += fabs(trA.getCell(i, j));
		}
		if (curMax > A1) A1 = curMax;
	}
	return A1;
}

float getAinf(Matrix& A) {
	float Ainf = 0;
	for (int i = 0; i < A.getN(); ++i) {
		float curMax = 0;
		for (int j = 0; j < A.getN(); ++j) {
			curMax += fabs(A.getCell(i, j));
		}
		if (curMax > Ainf) Ainf = curMax;
	}
	return Ainf;
}

Matrix createMatrixB(Matrix& A) {
	Matrix B(A.getN());
	Matrix At = A.transpose();
	float A1 = getA1(A);
	float Ainf = getAinf(A);
	B = At / (A1 * Ainf);

	return B;
}

Matrix createMatrixI(int N) {
	Matrix I(N);
	for (int i = 0; i < N; ++i) {
		for (int j = 0; j < N; ++j) {
			I.setCell(i, j, static_cast<float>(i == j));
		}
	}
	return I;
}
```

```
Matrix Matrix::invert(int M) {
	Matrix B = createMatrixB(*this);
	Matrix I = createMatrixI(B.getN());
	Matrix R = I - B * (*this);
	Matrix powR = R;

	Matrix result = I;
	for (int i = 0; i < M; ++i) {
		result += R;
		R = powR * R;
	}

	result = result * B;

	return result;
}

void SetRandValues(int N) {
	std::ofstream input;
	input.open("matrix.txt");
	if (input.is_open()) {
		for (int i = 0; i < N; ++i) {
			float value = (float)rand() / float(RAND_MAX % 1000);
			if (rand() % 2 == 0) {
				value *= -1.0;
			}
			input << std::round(value * 100)/100 << std::endl;
		}
	}
}

int main() {
	std::ifstream input;
	input.open("matrix.txt");
	if (input.is_open()) {
		Matrix A(SIZE);
		A.init(input);
		time_t start, end;
		time(&start);
		Matrix I = A.invert(10);
		time(&end);
		Matrix E = A * I;
		std::cout << "Time taken to find matrix I (inverted): " <<
difftime(end, start) << "sec\n";
		std::cout << "E = A * I\n";
		std::cout << "|E_1| = " << getA1(E) << std::endl;
		std::cout << "|E_inf| = " << getAinf(E) << std::endl;
	}

	return 0;
}
```

# Приложение 2. Перегруженный оператор сложения (intrinsic)

```
106  Matrix operator+(const Matrix& left, const Matrix& right) {
107      Matrix result(left.N);
108      __m128* m128_result = (__m128*)result.matrix;
109      __m128* m128_left = (__m128*)left.matrix;
110      __m128* m128_right = (__m128*)right.matrix;
111      for (int i = 0; i < left.N * left.N / 4; ++i) {
112          m128_result[i] = _mm_add_ps(m128_left[i], m128_right[i]);
113      }
114      return result;
115  }
```

## Приложение 3. Перегруженный оператор вычитания (intrinsic)

```
117 Matrix operator-(const Matrix& left, const Matrix& right) {
118     Matrix result(left.N);
119     __m128* m128_result = (__m128*)result.matrix;
120     __m128* m128_left = (__m128*)left.matrix;
121     __m128* m128_right = (__m128*)right.matrix;
122     for (int i = 0; i < left.N * left.N / 4; ++i) {
123         m128_result[i] = _mm_sub_ps(m128_left[i], m128_right[i]);
124     }
125     return result;
126 }
```

## Приложение 4. Перегруженный оператор умножения (intrinsic)

```
85  Matrix operator*(const Matrix& left, const Matrix& right) {
86      int N = left.N;
87      __m128* m128_right = (__m128*)right.matrix;
88      Matrix result(N);
89      __m128* m128_result = (__m128*)result.matrix;
90      __m128 mult;
91      __m128 tmp;
92
93      for (int i = 0; i < N; ++i) {
94          for (int j = 0; j < N; ++j) {
95              mult = _mm_set_ps1(left.matrix[i * N + j]);
96              for (int k = 0; k < N / 4; ++k) {
97                  tmp = _mm_mul_ps(mult, m128_right[N * j / 4 + k]);
98                  m128_result[N * i / 4 + k] = _mm_add_ps(m128_result[N * i / 4 + k], tmp);
99              }
100         }
101     }
102
103     return result;
104 }
```

## Приложение 5. Перегруженный оператор сложения (BLAS)

```
 99 Matrix operator+(const Matrix& left, const Matrix& right) {
100     int N = left.N;
101     Matrix result(N);
102     cblas_scopy(N*N, left.matrix, 1.0, result.matrix, 1.0);
103     cblas_saxpy(N*N, 1.0, right.matrix, 1.0, result.matrix, 1.0);
104     return result;
105 }
```

## Приложение 6. Перегруженный оператор вычитания (BLAS)

```
107 Matrix operator-(const Matrix& left, const Matrix& right) {
108     int N = left.N;
109     Matrix result(N);
110     cblas_scopy(N*N, left.matrix, 1.0, result.matrix, 1.0);
111     cblas_saxpy(N*N, -1.0, right.matrix, 1.0, result.matrix, 1.0);
112     return result;
113 }
```

## Приложение 7. Перегруженный оператор умножения (BLAS)

```
 92 Matrix operator*(const Matrix& left, const Matrix& right) {
 93     int N = left.N;
 94     Matrix result(N);
 95     cblas_sgemm(CblasRowMajor,
 96 <------><------>CblasNoTrans,
 97 <------><------>CblasNoTrans,
 98 <------><------>N,
 99 <------><------>N,
100 <------><------>N,
101 <------><------>1.0,
102 <------><------>left.matrix,
103 <------><------>N,
104 <------><------>right.matrix,
105 <------><------>N,
106 <------><------>0.0,
107 <------><------>result.matrix,
108 <------><------>N);
109
110     return result;
111 }
```