**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**отчет**

**по лабораторной работе №2**

**по дисциплине «Параллельные алгоритмы и системы»**

**Тема: «Оптимизация решения СЛАУ методом Гаусса»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9305 |  | Есин А.Ю. |
| Преподаватель |  | Пазников А.А. |

Санкт-Петербург

2023

Оглавление

[**Цель работы** 3](#_Toc131756665)

[**Основные положения** 3](#_Toc131756666)

[**Ход работы** 3](#_Toc131756667)

[**Сравнительный анализ** 4](#_Toc131756668)

[**Вывод** 8](#_Toc131756669)

# **Цель работы**

Оптимизировать метод решения систем линейных уравнений (СЛАУ) методом Гаусса.

# **Основные положения**

Матрица коэффициентов реализована с помощью std::vector. При этом для хранения данных используется вектор векторов типа double. Также с помощью вектора типа double реализован столбец свободных членов и выходная матрица.

При инициализации задается размер матрицы и столбца свободных членов.

Также был написать testbench, который обходит n векторов размерностью N (в нашем случае 1000 элементов) в прямом направлении и обратном направлении. При этом value внутри вектора лежит в диапазоне от 1 до 100.

# **Ход работы**

В первую очередь необходимо было найти hotspot. Это было сделано с помощью утилиты perf командами record и report. Результат поиска hotspot приведен на рисунке 1.

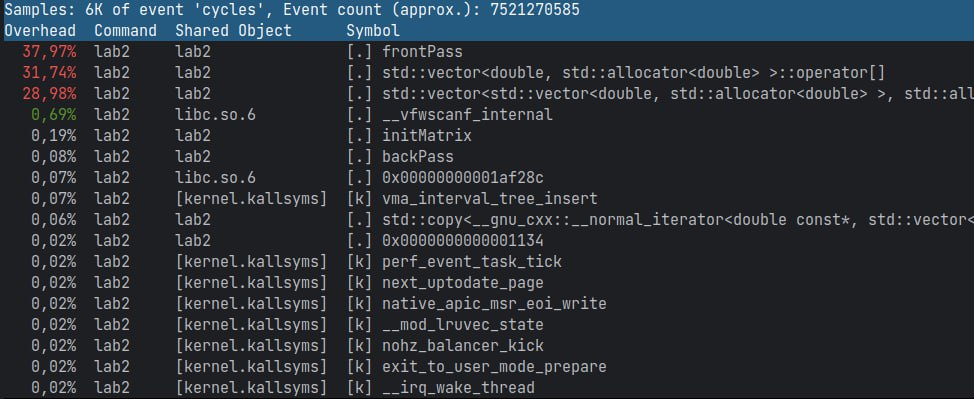


Рисунок 1. Отчетность утилиты perf

# **Сравнительный анализ**

Результаты сравнительного анализа приведены в таблице 1, а также представлены на рисунках 2-7

Таблица 1. Сравнительный анализ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Time elapsed | Cpu-migration | Cache-misses | Cache-references | Компилятор |
| Без оптимизации | 1,592 | 0 | 15,547% | 48930821 | gcc |
| Замена векторов на массивы | 0,609 | 0 | 10,775% | 49552171 | gcc |
| OpenMP + массивы | 0.462 | 0 | 10,988% | 36261542 | gcc |
| Без оптимизации | 1,658 | 0 | 15,243% | 49371337 | clang |
| Замена векторов на массивы | 0,554 | 1 | 12,188% | 49611946 | clang |
| OpenMP + массивы | 0,408 | 1 | 9,794% | 36570444 | clang |

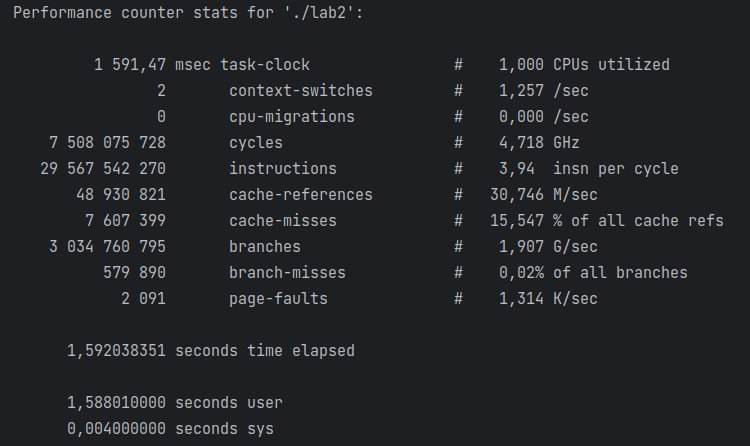


Рисунок 2. Без оптимизации g++

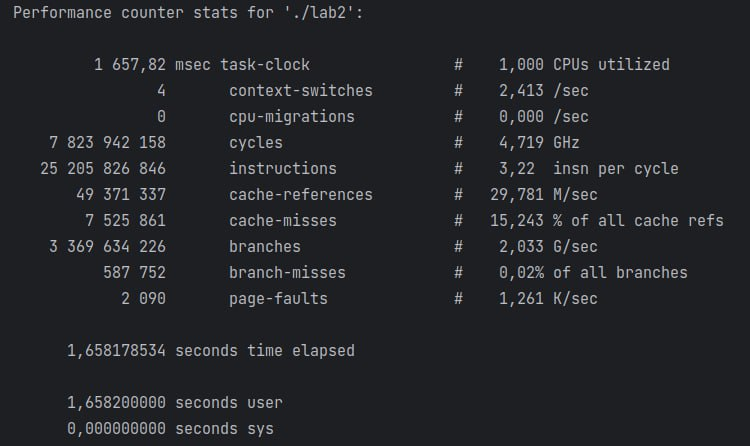


Рисунок 3. Без оптимизации clang

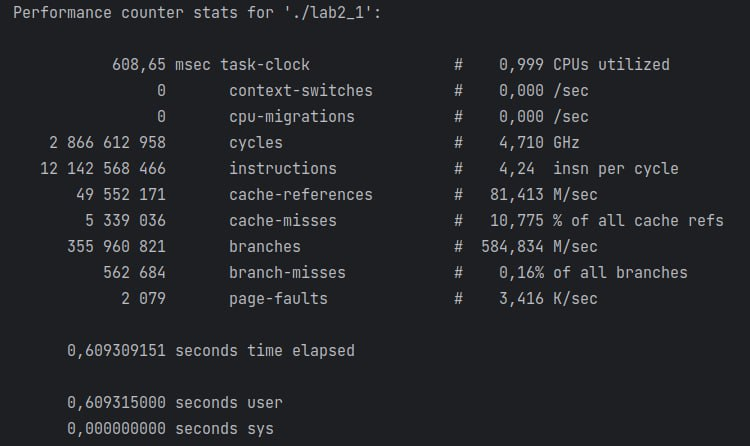


Рисунок 4. Замена векторов на массивы g++

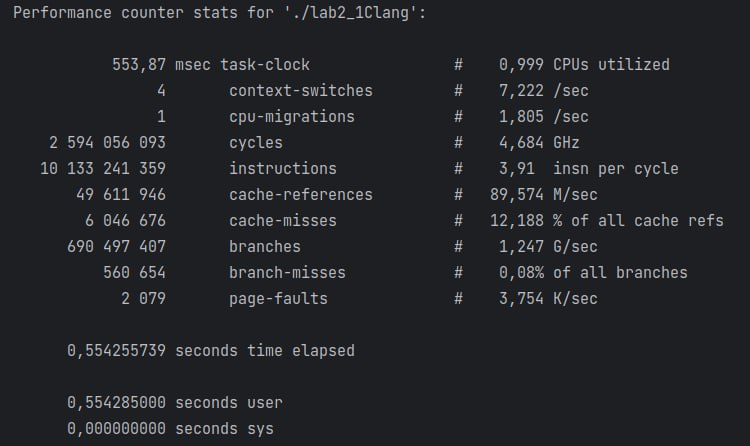


Рисунок 5. Замена векторов на массивы clang

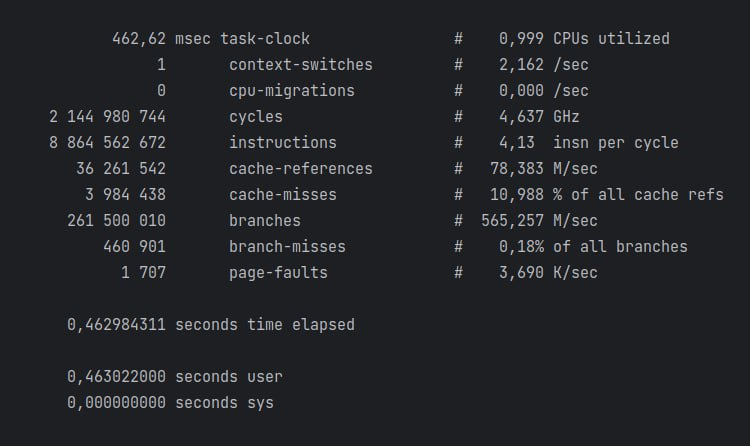


Рисунок 6. OpenMP + массивы g++

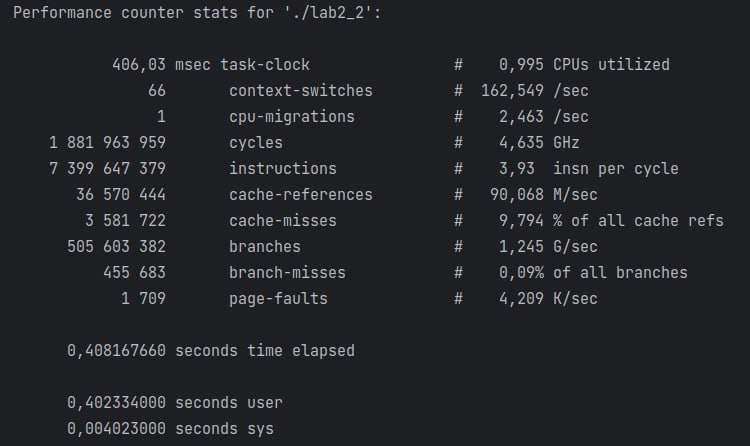


Рисунок 7. OpenMP + массивы clang

Для таблицы, приведенной выше, были использованы следующие метрики:

* Time elapsed – общее время выполнения записи с помощью perf
* Cpu-migration – количество переключений ядер процессора
* Cache-misses – количество обращений к кэшу, когда необходимых данных в нем нет
* Cache-references – общее количество обращений к кэшу

# **Вывод**

В ходе выполнения лабораторной работы был оптимизирован алгоритм решения систем линейных уравнений (СЛАУ) методом Гаусса

В ходе анализа hotspot’a было вявлено, что в оптимизации нуждаются метод frontPass, а также операции над std::vector.

По полученным данным мы увидели, что распараллеливание с помощью openMP и использование примитивных структур в С++ значительно повышает производительность вычислений.

**Текст программы**

Файл main.cpp для базового алгоритма.

#include <iostream>

#include <vector>

void initMatrix(std::vector<std::vector<double>>& A, std::vector<double>& B, int n) {

for (int i = 0; i < n; i++) { // Инициализация

srand(i \* (n + 1));

for (int j = 0; j < n; j++)

A[i][j] = rand() % 100 + 1;

B[i] = rand() % 100 + 1;

}

}

void frontPass(std::vector<std::vector<double>>& A, std::vector<double>& B, int n) {

// Прямой ход по всем уравнениям

for (int k = 0; k < n - 1; k++) {

// Исключение x\_k из строк k+1...n-1

double pivot = A[k][k];

for (int i = k + 1; i < n; i++) {

// Из уравнения (строки) i вычитается уравнение k

double lik = A[i][k] / pivot;

for (int j = k; j < n; j++)

A[i][j] -= lik \* A[k][j];

B[i] -= lik \* B[k];

}

}

}

void backPass(std::vector<std::vector<double>>& A, std::vector<double>& B, std::vector<double>& X, int n) {

// Обратный ход

for (int k = n - 1; k >= 0; k--) {

X[k] = B[k];

for (int i = k + 1; i < n; i++)

X[k] -= A[k][i] \* X[i];

X[k] /= A[k][k];

}

}

int main(int argc, char \*argv[]) {

int n = 1000; // Размерность матрицы

std::vector<std::vector<double>> A(n, std::vector<double>(n, 0)); // Матрица коэффициентов

std::vector<double> B(n); // Столбец свободных членов

std::vector<double> X(n); // Неизвестные

/\*

for (int i = 0; i < n; i++) { // Инициализация

srand(i \* (n + 1));

for (int j = 0; j < n; j++)

A[i][j] = rand() % 100 + 1;

B[i] = rand() % 100 + 1;

}\*/

initMatrix(A, B, n);

frontPass(A, B, n);

backPass(A, B, X, n);

return 0;

}

Файл main.cpp для алгоритма с заменой векторов на массивы.

#include <iostream>

int main(int argc, char \*argv[]) {

int n = 900; // Размерность матрицы

double A[n][n]; // Матрица коэффициентов

double B[n]; // Столбец свободных членов

double X[n]; // Неизвестные

for (int i = 0; i < n; i++) { // Инициализация

srand(i \* (n + 1));

for (int j = 0; j < n; j++)

A[i][j] = rand() % 100 + 1;

B[i] = rand() % 100 + 1;

}

// Прямой ход по всем уравнениям

for (int k = 0; k < n - 1; k++) {

// Исключение x\_k из строк k+1...n-1

double pivot = A[k][k];

for (int i = k + 1; i < n; i++) {

// Из уравнения (строки) i вычитается уравнение k

double lik = A[i][k] / pivot;

for (int j = k; j < n; j++)

A[i][j] -= lik \* A[k][j];

B[i] -= lik \* B[k];

}

}

// Обратный ход

for (int k = n - 1; k >= 0; k--) {

X[k] = B[k];

for (int i = k + 1; i < n; i++)

X[k] -= A[k][i] \* X[i];

X[k] /= A[k][k];

}

return 0;

}

Файл main.cpp для алгоритма с распараллеливанием.

#include <iostream>

#include <sys/time.h>

#include <iomanip>

#include <omp.h>

int main(int argc, char \*argv[]) {

int n = 3;

double \*a;

a = (double \*) malloc(sizeof(\*a) \* n \* n);

double \*b = (double \*) malloc(sizeof(\*b) \* n);

double \*x = (double \*) malloc(sizeof(\*x) \* n);

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

srand(i \* (n + 1));

for (int j = 0; j < n; j++)

a[i \* n + j] = rand() % 100 + 1;

b[i] = rand() % 100 + 1;

}

#pragma omp parallel

for (int k = 0; k < n - 1; k++) {

#pragma omp for

double pivot = a[k \* n + k];

for (int i = k + 1; i < n; i++) {

double lik = a[i \* n + k] / pivot;

for (int j = k; j < n; j++)

a[i \* n + j] -= lik \* a[k \* n + j];

b[i] -= lik \* b[k];

}

}

#pragma omp barrier

double s;

for (int k = n - 1; k >= 0; k--) {

s = 0;

#pragma omp for reduction(+:s)

for (int i = k + 1; i < n; i++)

s += a[k \* n + i] \* x[i];

#pragma omp single

x[k] = (b[k] - s) / a[k \* n + k];

}

free(b);

free(a);

free(x);

return 0;

}