**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 алгоритма перемножения матриц»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9307 |  | Николаев Н.В. |
| Преподаватель |  | Пазников А.А. |

Санкт-Петербург

2023

Оглавление

[**Цель работы** 3](#_Toc131752548)

[**Основные положения** 3](#_Toc131752549)

[**Измерения** 3](#_Toc131752550)

[**1.** **Без оптимизаций** 4](#_Toc131752551)

[**2.** **Распараллеливание вычисление с помощью openMP** 4](#_Toc131752552)

[**3.** **Использование стандартных массивов С вместо std::vector** 4](#_Toc131752553)

[**Вывод** 5](#_Toc131752554)

[**Приложение** 6](#_Toc131752555)

# **Цель работы**

Разработать алгоритм перемножения матриц и оптимизировать операцию перемножения.

# **Основные положения**

Было решено представить матрицу в виде двумерного массива. Под массивом в данном случае понимается примитив std::vector, т.к. он имеет удобный синтаксис добавления новых элементов.

При этом во внимание берется следующее допущение: количество строк у первой матрицы должно равняться количеству столбцов второй матрицы. Иначе алгоритм перемножения не сработает, т.к. элементы строк и столбцов перемножаются попарно, а затем складываются.

Также стоит принять во внимание тот факт, что если переставить матрицы местами, то должен получиться другой результат. А значит функция должна просчитывать цикл прохождения по элементов в зависимости от того, на какой позиции стоят её аргументы.

На рисунке 1 представлен пример перемножения матрицы 2х2 на матрицу 3х2:

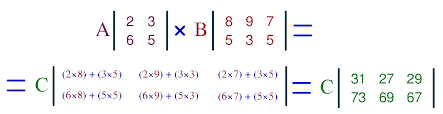


Рисунок 1. Пример перемножения матриц

# **Измерения**

Измерения буду проходить по заранее написанному тесту на 1000 элементов в матрице в каждой строке и каждом столбце. При этом внутри программы замеряется время выполнения, а с помощью утилиты perf в системе Linux будет произведен замер других характеристик, таких как кэш-миссы.

Для анализа быстродействия алгоритмы будут использоваться следующие параметры:

• Task-clock – время выполнения задачи

• Context-switch – количество переключений контекста выполнения

• Cache-misses – количество обращений к кэшу, когда необходимых данных в нем нет

• Branch-misses – количество неправильных предсказаний условных переходов

## **Без оптимизаций**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компилятор | Task-clock | Context-switches | Cache-misses | Branch-misses |
| GCC | 7327.87 | 2185 | 71.246% | 0.01% |
| CLANG | 8040.07 | 19 | 73.596% | 0.01% |

## **Распараллеливание вычисление с помощью openMP**

Одним из методов оптимизации является распараллеливание вычислений с помощью библиотеки openMP. У нас как раз имеется функция, которая имеет цикл, имеющий сложность О(n^2). Можно его распараллелить, и как видно из таблицы ниже, это сэкономит несколько секунд и context-switches.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компилятор | Task-clock | Context-switches | Cache-misses | Branch-misses |
| GCC | 6529.15 | 896 | 76.346% | 0.01% |
| CLANG | 6137.62 | 34 | 73.063% | 0.01% |

## **Использование стандартных массивов С вместо std::vector**

Отказ от использования сложных структур приводит к усложнению управлению данными, однако к упрощению выделения константного блока памяти. Использование встроенных примитивов внутри С++ можно добиться еще большей производительности, чем с помощью распараллеливания.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компилятор | Task-clock | Context-switches | Cache-misses | Branch-misses |
| GCC | 2,69 | 3 | 71.797% | 2.30% |
| CLANG | 0,68 | 2 | 45,0541% | 2.30% |

# **Вывод**

В ходе выполнения лабораторной работы был оптимизирован алгоритм операции перемножения матриц.

Оригинальный алгоритм показал, что метрика Cache-misses, Task-clock и Context-switch можно оптимизировать

Оба метода оптимизации хорошо сказались на производительности, однако наиболее действенным оказалось замена сложных структур типа std::vector на более примитивные в языке С++.

# **Приложение**

Файл main.cpp для базового алгоритма.

#include <iostream>  
#include <vector>  
#include <chrono>  
  
*// Функция для перемножения матриц A и B*std::vector<std::vector<int>> multiplyMatrices(const std::vector<std::vector<int>>& A, const std::vector<std::vector<int>>& B) {  
 int M = A.size(); *// Количество строк в матрице A* int N = A[0].size(); *// Количество столбцов в матрице A* int P = B[0].size(); *// Количество столбцов в матрице B  
  
 // Создаем результирующую матрицу C и заполняем ее нулями* std::vector<std::vector<int>> C(M, std::vector<int>(P, 0));  
  
 *// Перемножаем матрицы A и B* for (int i = 0; i < M; i++) {  
 for (int j = 0; j < P; j++) {  
 for (int k = 0; k < N; k++) {  
 C[i][j] += A[i][k] \* B[k][j];  
 }  
 }  
 }  
  
 return C;  
}  
  
int main() {  
 *// Пример входных данных* int size = 1000;  
 std::vector<std::vector<int>> A = {};  
 std::vector<std::vector<int>> B = {};  
  
 for (int i = 0; i < size; ++i){  
 std::vector<int> currentRowA;  
 std::vector<int> currentRowB;  
 A.push\_back(currentRowA);  
 B.push\_back(currentRowB);  
 for (int j = 0; j < size; ++j){  
 A[i].push\_back(size\*i+j);  
 B[i].push\_back(size\*(i+1)-j);  
 }  
 }  
  
  
  
 *// Перемножаем матрицы A и B* auto start = std::chrono::high\_resolution\_clock::now();  
 std::vector<std::vector<int>> C = multiplyMatrices(A, B);  
 auto stop = std::chrono::high\_resolution\_clock::now();  
 auto testTime = std::chrono::duration\_cast<std::chrono::duration<double>>(stop - start);  
 std::cout << "Test time: " << testTime.count() << "seconds" << std::endl;  
  
 return 0;  
}

Файл main.cpp для алгоритма с распараллеливание openMP

#include <iostream>  
#include <vector>  
#include <chrono>  
  
*// Функция для перемножения матриц A и B*std::vector<std::vector<int>> multiplyMatrices(const std::vector<std::vector<int>>& A, const std::vector<std::vector<int>>& B) {  
 int M = A.size(); *// Количество строк в матрице A* int N = A[0].size(); *// Количество столбцов в матрице A* int P = B[0].size(); *// Количество столбцов в матрице B  
  
 // Создаем результирующую матрицу C и заполняем ее нулями* std::vector<std::vector<int>> C(M, std::vector<int>(P, 0));  
  
 *// Перемножаем матрицы A и B*#pragma omp parallel for schedule(static)  
 for (int i = 0; i < M; i++) {  
 for (int j = 0; j < P; j++) {  
 for (int k = 0; k < N; k++) {  
 C[i][j] += A[i][k] \* B[k][j];  
 }  
 }  
 }  
  
 return C;  
}  
  
int main() {  
 *// Пример входных данных* int size = 1000;  
 std::vector<std::vector<int>> A = {};  
 std::vector<std::vector<int>> B = {};  
  
 for (int i = 0; i < size; ++i){  
 std::vector<int> currentRowA;  
 std::vector<int> currentRowB;  
 A.push\_back(currentRowA);  
 B.push\_back(currentRowB);  
 for (int j = 0; j < size; ++j){  
 A[i].push\_back(size\*i+j);  
 B[i].push\_back(size\*(i+1)-j);  
 }  
 }  
  
  
  
 *// Перемножаем матрицы A и B* auto start = std::chrono::high\_resolution\_clock::now();  
 std::vector<std::vector<int>> C = multiplyMatrices(A, B);  
 auto stop = std::chrono::high\_resolution\_clock::now();  
 auto testTime = std::chrono::duration\_cast<std::chrono::duration<double>>(stop - start);  
 std::cout << "Test time: " << testTime.count() << "seconds" << std::endl;  
  
 return 0;  
}

Файл main.cpp для алгоритма с заменой векторов на примитивные средства С++

#include <iostream>  
#include <chrono>  
  
const int size = 1000;  
  
*// Функция для перемножения матриц A и B*void multiplyMatrices(int A[size][size], int B[size][size], int C[size][size]) {  
 for (int i = 0; i < 3; ++i) {  
 for (int j = 0; j < 3; ++j) {  
 C[i][j] = 0;  
 for (int k = 0; k < 3; ++k) {  
 C[i][j] += A[i][k] \* B[k][j];  
 }  
 }  
 }  
}  
  
int main() {  
 *// Пример входных данных* int A[size][size];  
 int B[size][size];  
 int C[size][size];  
  
 for (int i = 0; i < size; ++i) {  
 for (int j = 0; j < size; ++j) {  
 A[i][j] = i + j;  
 B[i][j] = i + j + 2;  
 }  
 }  
  
  
  
 *// Перемножаем матрицы A и B* auto start = std::chrono::high\_resolution\_clock::now();  
 multiplyMatrices(A, B, C);  
 auto stop = std::chrono::high\_resolution\_clock::now();  
 auto testTime = std::chrono::duration\_cast<std::chrono::duration<double>>(stop - start);  
 std::cout << "Test time: " << testTime.count() << "seconds" << std::endl;  
  
 return 0;  
}