ОТЧЕТ

По лабораторной работе №4

По курсу «Анализ алгоритмов»

Тема: «Распараллеливание вычислений»

|  |  |
| --- | --- |
|  | **Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  «Московский государственный технический университет  имени Н.Э. Баумана  (национальный исследовательский университет)»  (МГТУ им. Н.Э. Баумана) |

Студент: Зейналов З. Г.

Группа: ИУ7-51

Преподаватель: Волкова Л.Л.

Москва, 2019г.

Оглавление

[Введение 3](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736334)

[Задачи работы: 4](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736335)

[1. Аналитическая часть 5](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736336)

[1.1 Описание алгоритмов 5](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736337)

[1.2 Алгоритм умножения Копперсмита-Винограда 6](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736338)

[2. Конструкторская часть 7](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736339)

[2.1 Разработка алгоритмов 8](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736340)

[2.2 Расчет трудоемкости 14](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736341)

[3 Технологическая часть 15](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736342)

[3.1 Требования к программному обеспечению 15](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736343)

[3.2 Средства реализации 15](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736344)

[3.3 Листинг кода 16](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736345)

[4 Экспериментальная часть 19](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736346)

[4.1 Примеры работы 19](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736347)

[4.2 Результаты тестирования 20](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736348)

[4.3 Постановка эксперимента по замеру времени 20](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736349)

[4.4 Сравнительный анализ на материале экспериментальных данных 21](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736350)

[Заключение 23](file:///C:\Users\зейнал\Desktop\отчет%20по%20АА%202.docx#_Toc22736351)

# Введение

**Целью** данной лабораторной работы является исследование существующих алгоритмов умножения матриц и трудоемкости их вычисления.

# Задачи работы:

Задачами данной лабораторной работы являются:

1. Научиться писать многопоточные программы;
2. Применить полученные знания на практике с использованием алгоритма Копперсмита-Винограда в несколько потоков.
3. Провести замеры времени работы однопоточной и многопоточной реализаций и проанализировать полученные данные.
4. **Аналитическая часть**

В данном разделе будут представлены описания алгоритмов, формулы и оценки сложностей алгоритмов

### Алгоритм умножения Копперсмита-Винограда

Алгоритм Копперсмита—Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом (англ.). В исходной версии асимптотическая сложность алгоритма составляла , где n — размер стороны матрицы. Алгоритм Копперсмита—Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нeм представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим 2 вектора:

Их скалярное произведение равно:

Что эквивалентно

Не очевидным остается тот факт, что выражение в правой части формулы 2 допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого стобца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие 5 сложений, а также два сложения.

## Конструкторская часть

В данном разделе будут размещены схемы алгоритмов.

### Разработка алгоритмов

На рисунках 1 – 5 приведены схемы алгоритма, демонстрирующего работу.

Рисунок 1. Оптимизированный алгоритм умножения Винограда (часть 1)



Рисунок 2. Оптимизированный алгоритм умножения Винограда (часть 2)



Рисунок 3. Оптимизированный алгоритм умножения Винограда (часть 3)



Рисунок 4. Оптимизированный алгоритм умножения Винограда (часть 4)



Рисунок 5. Оптимизированный алгоритм умножения Винограда (часть 5)

## Технологическая часть

В данном разделе будут приведены Требования к программному обеспечению, средства реализации, листинг кода и примеры тестирования.

### Средства реализации

В качестве языка программирования был выбран С++ (компилятор g++) в связи с его широким функционалом и быстротой работы, а так же благодаря привычному для меня синтаксису и семантики языка. Среда разработки - Qt. Для работы с потоками использовалась библиотека thread. Время работы процессора замеряется с помощью функции std::chrono().

### Листинг кода

В листинге 1 представлена однопоточная реализация. Однако для многопоточной реализации алгоритма, функция была разбита на 3 части, представленные в листингах 2 – 4.

int **optimizedMultiplication**(int \*\*\*result, int &rc, int &cc, int\*\* src1, int \*\*src2, int rc1, int cc1, int rc2, int cc2)

{

*if* (cc1 != rc2)

{

*return* ERR\_MTR\_SIZE;

}

*if* (src1 == NULL || src2 == NULL)

{

*return* ERR\_EMPTY\_MATRIX;

}

int rc2\_2 = rc2 >> 1; // opti

rc = rc1;

cc = cc2;

int \*\*tmpMtr = allocateMatrix(rc, cc);

int \*MulH = (int\*)calloc(rc1, *sizeof*(int));

int \*MulV = (int\*)calloc(cc2, *sizeof*(int));

*for* (int i = 0; i < rc1; i++)

{

*for* (int j = 0; j < rc2\_2; j++) // opti

{

MulH[i] -= src1[i][j \* 2] \* src1[i][2 \* j + 1]; // opti

}

}

*for* (int i = 0; i < cc2; i++)

{

*for* (int j = 0; j < rc2\_2; j++) // opti

{

MulV[i] -= src2[j \* 2][i] \* src2[2 \* j + 1][i]; // opti

}

}

int N = rc2 - 1; // opti

bool flag = rc2 % 2; // opti

*for* (int i = 0; i < rc1; i++)

{

*for* (int j = 0; j < cc2; j++)

{

tmpMtr[i][j] = MulH[i] + MulV[j];

*for* (int k = 0; k < rc2\_2; k++) // opti

{

tmpMtr[i][j] += (src1[i][k \* 2] + src2[2 \* k + 1][j]) \* (src1[i][2 \* k + 1] + src2[2 \* k][j]); // opti

}

}

}

*for* (int i = 0; i < rc1; i++)

{

*for* (int j = 0; j < cc2; j++)

{

*if* (flag) // opti

{

tmpMtr[i][j] += src1[i][N] \* src2[N][j]; // opti

}

}

}

\*result = tmpMtr;

}

Листинг 1. Реализация оптимизированного Алгоритма Винограда

void **second\_part**(Matrix src2, int columns\_begin, int cc2, int rc2\_2, std::vector<int> &MulV)

{

*for* (int i = columns\_begin; i < cc2; i++)

{

*for* (int j = 0; j < rc2\_2; j++)

{

MulV[i] -= src2.matrix[j \* 2][i] \* src2.matrix[2 \* j + 1][i];

}

}

}

Листинг 3 - Часть вычисления вектора столбца

Листинг 1 - Часть вычисления вектора строки

void **first\_part**(Matrix src1, int rows\_begin ,int rc1, int rc2\_2, std::vector<int> &MulH)

{

*for* (int i = rows\_begin; i < rc1; i++)

{

*for* (int j = 0; j < rc2\_2; j++)

{

MulH[i] -= src1.matrix[i][j \* 2] \* src1.matrix[i][2 \* j + 1];

}

}

}

Листинг 4 -- Часть перемножения

void **last\_part**(Matrix src1, Matrix src2, std::vector<int> &MulH, std::vector<int> &MulV, int rows\_begin, int rc1, int cc2, int rc2\_2, Matrix &mtr\_res)

{

int N = src2.rows\_count - 1;

bool flag = src2.rows\_count % 2;

int rc = rc1;

int cc = cc2;

mtr\_res.rows\_count = rc1;

mtr\_res.columns\_count = cc2;

int \*\*tmpMtr = allocateMatrix(rc, cc);

*for* (int i = 0; i < rc1; i++)

{

*for* (int j = 0; j < cc2; j++)

{

tmpMtr[i][j] = MulH[i] + MulV[j];

*for* (int k = 0; k < rc2\_2; k++)

{

tmpMtr[i][j] += (src1.matrix[i][k \* 2] + src2.matrix[2 \* k + 1][j]) \* (src1.matrix[i][2 \* k + 1] + src2.matrix[2 \* k][j]);

}

}

}

*for* (int i = 0; i < rc1; i++)

{

*for* (int j = 0; j < cc2; j++)

{

*if* (flag)

{

tmpMtr[i][j] += src1.matrix[i][N] \* src2.matrix[N][j];

}

}

}

mtr\_res.matrix = tmpMtr;

}

В листинге 5 представлена двупоточная реализация алгоритма. Реализации для 4, 8, 16 потоков имеют аналогичную реализацию.

int **multi2**(Matrix mtr1, Matrix mtr2, Matrix &mtr\_res)

{

*if* (mtr1.columns\_count != mtr2.rows\_count)

{

*return* CANT\_MULTIPLY;

}

int rows\_count\_first = mtr1.rows\_count;

int rows\_count\_second\_2 = mtr2.rows\_count / 2;

int columns\_count\_second = mtr2.columns\_count;

mutex m1rc, m2rc\_2, m2cc;

mutex m1, m2, m\_out;

mutex row, column;

mutex outout;

std::vector<int> MulH(mtr1.rows\_count);

std::vector<int> MulV(mtr2.columns\_count);

thread thread\_1\_1(first\_part, mtr1, 0,

rows\_count\_first,

rows\_count\_second\_2,

ref(MulH));

thread thread\_2\_1(second\_part, mtr2, 0,

columns\_count\_second,

rows\_count\_second\_2,

ref(MulV));

*if* (thread\_1\_1.joinable() && thread\_2\_1.joinable())

{

thread\_1\_1.join();

thread\_2\_1.join();

}

thread thread\_3\_1(last\_part, mtr1,

mtr2,

ref(MulH),

ref(MulV), 0,

rows\_count\_first / 2,

columns\_count\_second,

rows\_count\_second\_2,

ref(mtr\_res));

thread thread\_3\_2(last\_part, mtr1,

mtr2,

ref(MulH),

ref(MulV), rows\_count\_first / 2,

rows\_count\_first,

columns\_count\_second,

rows\_count\_second\_2,

ref(mtr\_res));

thread\_3\_1.join();

thread\_3\_2.join();

}

Листинг 5 – двупоточная реализация алгоритма.

## Экспериментальная часть

В данном разделе будут приведены постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных. Анализ производился на персональном компьютере с процессором с 4 ядрами, 4 потоками.

### Постановка эксперимента по замеру времени

Для произведения замеров времени выполнения реализаций алгоритмов будет использована следующая формула , где t – время выполнения, N – количество замеров. Неоднократное измерение времени необходимо для построения более гладкого графика.

Количество замеров будет взято равным 100.

Тестирование будет проведено на одинаковых входных данных. 1) Матрицы размерностями от 100х100 до 500х500 с шагом 100.

### Сравнительный анализ на материале экспериментальных данных

Ниже приведены график зависимости временных затрат от размеров входных данных.

Вывод

# Заключение

В ходе работы были изучены и реализованы алгоритмы стандартного умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Был проведен сравнительный анализ перечисленных алгоритмов по трудоемкости и экспериментально выявлена временная разница работы алгоритмов. Классический алгоритм в неоптимизированном виде является более эффективным чем алгоритм винограда, однако после ряда оптимизаций, алгоритм Винограда становится значительно быстрее классического.