

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук, Департамент программной инженерии
Дисциплина: «Архитектура вычислительных систем»

Вариант 1

**МНОГОПОТОЧНАЯ ПРОГРАММА
ВЫЧИСЛЕНИЯ ПРОИЗВЕДЕНИЯ МАТРИЦ**

Пояснительная записка

Выполнила:
Абу Аль Лабан Надя,
студент гр. БПИ198.

Москва
2020

Содержание

1. Текст задания.....	3
2. Применяемые расчетные методы	4
2.1. Теория решения задания	4
2.2. Дополнительный функционал программы	5
3. Тестирование программы.....	6
3.1. Корректные значения	6
3.2. Некорректные значения	9
Список литературы.....	10
Код программы	11

1. Текст задания

Вычислить векторное произведение квадратных матриц A и B . Входные данные: произвольные квадратные матрицы A и B одинаковой размерности. Размер матриц задается входным параметром. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

2. Применяемые расчетные методы

2.1. Теория решения задания

Матрицей называют совокупность элементов, расположенных в виде таблицы из m строк и n столбцов. Числа m и n называют размерами матрицы. В задании требуется найти произведение квадратных матриц, значит $m = n$.

Рассмотрим матрицу A с элементами a_{ij} ($i = 1..n$, $j = 1..n$) и матрицу B с элементами b_{ke} ($k = 1..n$, $e = 1..n$). Произведением A и B называют матрицу C с элементами $c_{ij} = \sum_{r=1}^n a_{ir}b_{rj}$.

То есть для получения элемента c_{ij} результирующей матрицы, нужно скалярно умножить i -ю строку первой матрицы на j -й столбец второй:

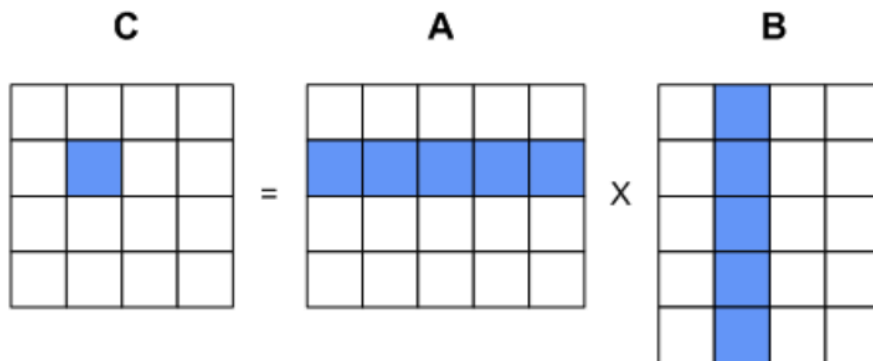


Рисунок 1. Схема умножения матриц.

Поскольку получения строк результирующей матрицы — независимые действия, требующие прохождение некоторых циклов, наиболее подходящая модель построения многопоточного приложения — итеративный параллелизм.

Итеративный параллелизм используется для реализации нескольких потоков (часто идентичных), каждый из которых содержит циклы. Потоки программы, описываются итеративными функциями и работают совместно над решением одной задачи.

Количество потоков задается с клавиатуры и записывается в переменную `threads_count`. Далее нужно как-то распределить строки матрицы между потоками. Для простоты все потоки кроме последнего будут обрабатывать ровно $n / \text{threads_count}$ строк, а последний обработает столько же и все оставшиеся в добавок.

Для запуска потока мы передаем ему структуру `thread_args`, содержащую в себе все необходимые для работы потока аргументы (матрицы, которые участвуют в умножении, и границы строк, за которые отвечает поток). Поток запускает функцию `thread_func`, которая в свою очередь запускает метод матрицы `c` — `thread_func`, который осуществляет вычисление строчек от номера `from` до номера `to`.

2.2. Дополнительный функционал программы

Пользователь вводит размерность матрицы и количество потоков. Поскольку оба числа должны быть положительными и целыми, при попытке ввести число меньше 1 или строку программа сообщает пользователю о некорректном вводе и запрашивает повторный ввод.

Также программа самостоятельно случайным образом генерирует матрицы с числами в заданном с помощью констант `MIN_VAL = 0` и `MAX_VAL = 10` диапазоне. Такие границы выбраны для простоты вычислений.

3. Тестирование программы

При запуске программы выводится просьба ввести размерность матриц (рис. 2).

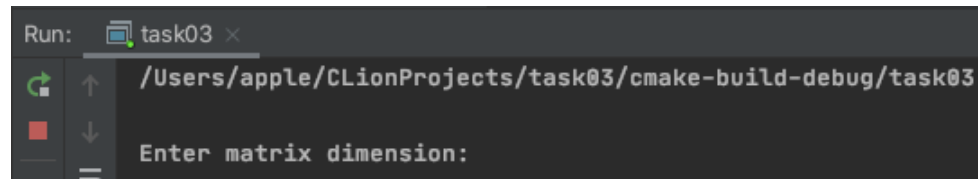


Рисунок 2. Запуск

3.1. Корректные значения

Для начала введем корректные значения и сверимся с матричным калькулятором [2] для проверки правильности работы алгоритма.

Начнем с граничного значения и введем 1 и для размерности, и для количества потоков (рис. 3). Матрица размерностью 1 представляет собой число, значит произведение матриц размерности 1 равно произведению чисел. Действительно, $1 * 3 = 3$. Результат вычислений верный.

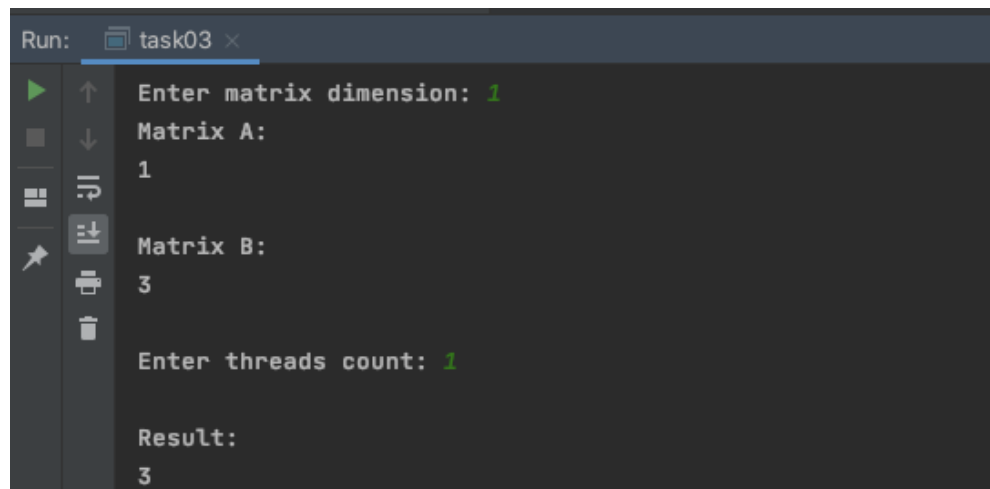


Рисунок 3. Ввод 1 в обоих случаях.

Введем значение для размерности 5 и количество потоков меньше 5, например, 3 (рис. 4). Результат верный (рис. 5).

```
Run: task03 x
/Users/apple/CLionProjects/task03/cmake-build-debug/task03

Enter matrix dimension: 5
Matrix A:
7 1 5 5 6
7 9 9 6 6
3 4 0 5 0
0 6 2 6 9
9 8 8 7 3

Matrix B:
5 3 9 0 2
9 7 0 6 8
7 9 1 5 1
9 7 6 3 9
5 8 7 7 1

Enter threads count: 3

Result:
154 156 140 88 78
263 255 150 159 155
96 72 57 39 83
167 174 101 127 113
251 228 152 130 156

Process finished with exit code 0
```

Рисунок 4. Ввод количества потоков меньшего, чем размерность.

$$\begin{pmatrix} 7 & 1 & 5 & 5 & 6 \\ 7 & 9 & 9 & 6 & 6 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 2 & 6 & 9 \\ 9 & 8 & 8 & 7 & 3 \end{pmatrix} \cdot \begin{pmatrix} 5 & 3 & 9 & 0 & 2 \\ 9 & 7 & 0 & 6 & 8 \\ 7 & 9 & 1 & 5 & 1 \\ 9 & 7 & 6 & 3 & 9 \\ 5 & 8 & 7 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 154 & 156 & 140 & 88 & 78 \\ 263 & 255 & 150 & 159 & 155 \\ 96 & 72 & 57 & 39 & 83 \\ 167 & 174 & 101 & 127 & 113 \\ 251 & 228 & 152 & 130 & 156 \end{pmatrix}$$

Рисунок 5. Подсчет результата второго примера на калькуляторе.

Теперь введем значение для размерности 5 и количество потоков большее 5, например, 7 (рис. 6). Результат верный (рис. 7).

```

Run: task03 x
/Users/apple/CLionProjects/task03/cmake-build-debug/task03

Enter matrix dimension: 5
Matrix A:
4 8 3 6 2
1 7 2 1 6
6 7 7 0 5
2 1 5 6 0
8 3 1 1 6

Matrix B:
7 0 0 2 1
8 9 8 7 5
5 6 2 6 7
4 7 8 1 7
3 4 1 8 9

Enter threads count: 7

Result:
137 140 120 104 125
95 106 74 112 111
148 125 75 143 135
71 81 66 47 84
107 64 40 92 91

Process finished with exit code 0

```

Рисунок 6. Ввод количества потоков большего, чем размерность.

$$\begin{pmatrix} 4 & 8 & 3 & 6 & 2 \\ 1 & 7 & 2 & 1 & 6 \\ 6 & 7 & 7 & 0 & 5 \\ 2 & 1 & 5 & 6 & 0 \\ 8 & 3 & 1 & 1 & 6 \end{pmatrix} \cdot \begin{pmatrix} 7 & 0 & 0 & 2 & 1 \\ 8 & 9 & 8 & 7 & 5 \\ 5 & 6 & 2 & 6 & 7 \\ 4 & 7 & 8 & 1 & 7 \\ 3 & 4 & 1 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 137 & 140 & 120 & 104 & 125 \\ 95 & 106 & 74 & 112 & 111 \\ 148 & 125 & 75 & 143 & 135 \\ 71 & 81 & 66 & 47 & 84 \\ 107 & 64 & 40 & 92 & 91 \end{pmatrix}$$

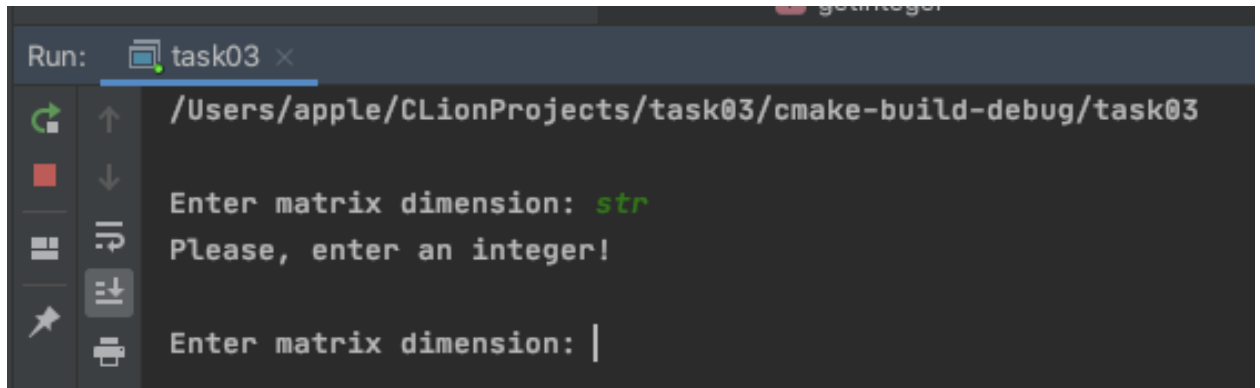
Рисунок 7. Подсчет результата третьего примера на калькуляторе.

Таким образом, мы показали, что программа работает корректно для граничных значений, а также как для меньшего, так и для большего, чем размерность матриц, количества потоков.

3.2. Некорректные значения

Чтобы проверить работу программы при некорректном вводе, нужно понять, какие случаи программа должна обрабатывать.

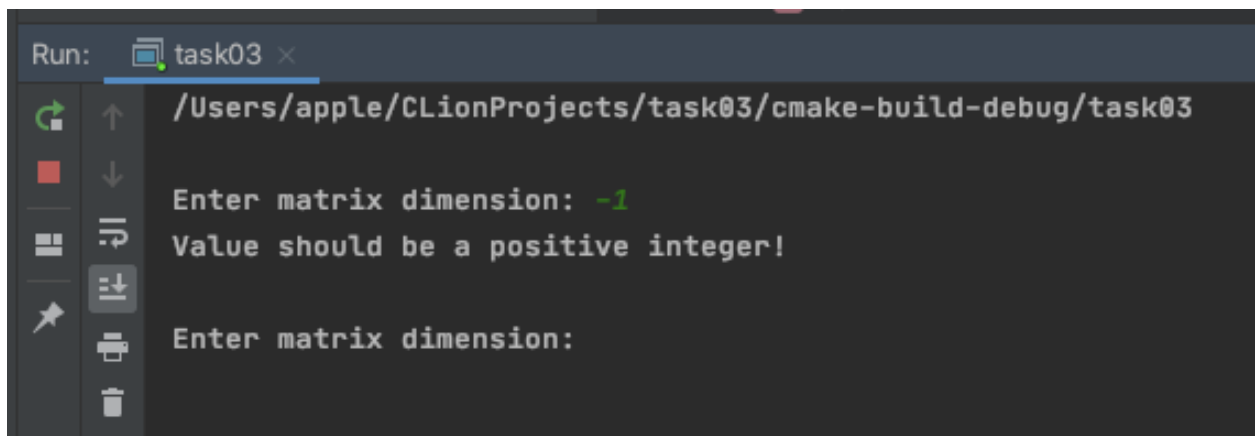
Для начала введем строковое значение, поскольку от пользователя требуется ввести число (рис. 8).



```
Run: task03 x
/Users/apple/CLionProjects/task03/cmake-build-debug/task03
Enter matrix dimension: str
Please, enter an integer!
Enter matrix dimension: |
```

Рисунок 8. Ввод строкового значения

Теперь введем значение меньшее, чем ограничение (рис. 9).



```
Run: task03 x
/Users/apple/CLionProjects/task03/cmake-build-debug/task03
Enter matrix dimension: -1
Value should be a positive integer!
Enter matrix dimension:
```

Рисунок 9. Ввод числа, не входящего в допустимые значения

Таким образом, мы показали, что программа обрабатывает некорректные данные и запрашивает повторный ввод.

Список литературы

1. Умножение матриц. [Электронный ресурс] // Режим доступа: свободный, URL: https://ru.wikipedia.org/wiki/Умножение_матриц (дата обращения: 15.11.2020)
2. Матричный калькулятор. [Электронный ресурс] // Режим доступа: свободный, URL: <https://matrixcalc.org> (дата обращения: 15.11.2020)
3. Руководство по FASM [Электронный ресурс]. // Режим доступа: свободный, URL: <http://flatassembler.narod.ru/fasm.htm> (дата обращения: 15.11.2020)

Код программы

```

#include <iostream>

#include <ctime>

#include<pthread.h>

#include<mutex>

class Matrix {
public:
    /**
     * Инициализирует квадратную матрицу заданного размера
     * и заполняет ее в диапазоне от MIN_VAL до MAX_VAL
     * @param size - размер матрицы
     */
    explicit Matrix(int size, bool empty = false) {
        this->size = size;
        matrix = new double* [size];
        for (int i = 0; i < size; ++i) {
            matrix[i] = new double [size];
            for (int j = 0; j < size; ++j) {
                matrix[i][j] = empty ? 0 : static_cast<double>(rand() % MAX_VAL + MIN_VAL);
            }
        }
    }
    /**
     * Считает k-ю строку в новой матрице
     * @param a - первая матрица в умножении
     * @param b - вторая матрица в умножении
     * @param k - номер строки
     */
    void mul_rows(Matrix* a, Matrix* b, int from, int to) {
        for (int k = from; k < to; ++k) {
            for (int i = 0; i < size; ++i) {
                for (int j = 0; j < size; ++j) {
                    matrix[k][i] += a->matrix[k][j] * b->matrix[j][i];
                }
            }
        }
    }
}

```

```

/**
 * Выводит матрицу в консоль
 */
void display() {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j)
            std::cout << matrix[i][j] << "\t";
        std::cout << "\n";
    }
}

~Matrix() {
    for (int i = 0; i < size; ++i) {
        delete [] matrix[i];
    }
}

private:
    const int MAX_VAL = 10;          // Максимальная граница рандома.
    const int MIN_VAL = 0;          // Минимальная граница рандома.
    int size;                        // Размерность матрицы.
    double **matrix;                 // Матрица.
};

struct thread_args {
    Matrix *a;                       // Первая матрица в умножении.
    Matrix *b;                       // Вторая матрица в умножении.
    Matrix *c;                       // Результирующая матрица.
    int from;                        // Строка с которой поток начинается.
    int to;                          // Строка, на которой поток заканчивает.
};

/**
 * Функция для потока.
 * @param arg - Аргументы потока.
 */
void* thread_func(void *arg) {
    auto *pArgs = (thread_args*) arg;

    pArgs->c->mul_rows(pArgs->a, pArgs->b, pArgs->from, pArgs->to);

    return nullptr;
}

```

```

}

/**
 * Считывает положительное целочисленное значение с клавиатуры
 * @param valName – наименование значения
 * @return целочисленное значение
 */
int getInteger(const std::string& valName) {
    int n;
    do {
        std::cout << "\nEnter " << valName << ": ";
        std::cin >> n;

        if (std::cin.fail())
        {
            std::cout << "Please, enter an integer!\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }

        if (n <= 0) {
            std::cout << "Value should be a positive integer! \n";
        }
    } while (n <= 0);
    return n;
}

int main() {
    std::srand(time(nullptr));

    // Запрашиваем размерность матрицы.
    int size = getInteger("matrix dimension");

    // Инициализируем и выводим матрицы.
    auto a = new Matrix(size);
    auto b = new Matrix(size);
    std::cout << "Matrix A:\n";
    a->display();
    std::cout << "\nMatrix B:\n";

```

```

b->display();

// Матрица для результата.
auto c = new Matrix(size, true);

// Запрашиваем количество потоков.
int threads_count = getInteger("threads count");

// Создаем массив потоков и аргументов к ним.
thread_args *args;
pthread_t *threads;

args = (thread_args*) malloc(sizeof(thread_args) * threads_count);
threads = (pthread_t*) malloc(sizeof(pthread_t) * threads_count);
// Считаем количество строк, обрабатываемых одним потоком.
int rows_per_thread = size / threads_count;
int residue = size % threads_count;
// Инициализируем аргументы для потоков.
for (int i = 0; i < threads_count; i++) {
    args[i].a = a;
    args[i].b = b;
    args[i].c = c;
    args[i].from = rows_per_thread * i;
    args[i].to = (i + 1)*rows_per_thread;
}
args[threads_count - 1].to += residue;
// Создаем потоки.
for (int i = 0; i < threads_count; i++) {
    pthread_create(&threads[i], nullptr, thread_func, (void*)&args[i]);
}
// Запускаем потоки.
for (int i = 0; i < threads_count; i++) {
    pthread_join(threads[i], nullptr);
}
// Выводим результат.
std::cout << "\nResult:\n";
c->display();

return 0;
}

```