

Федеральное государственное автономное образовательное учреждение высшего образования «ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» Факультет математики, механики и компьютерных наук Кафедра математического моделирования Направление подготовки 010501 —«Прикладная математика и информатика»

**Индивидуальная работа на тему: «Параллельное блочное
умножение матриц»**

Студент 4 курса: К. В. Бриних

Ростов-на-Дону 2018

Оглавление

1. Блочное умножение матриц	
1.1. Постановка задачи	3
1.2. Метод решения	3
2. Хранение матриц	
2.1. Матрица A	4
2.2. Матрица B.	5
3. Сравнение результатов	

Глава 1

Блочное умножение матриц

1.1 Постановка задачи

Написать программу блочного умножения двух матриц $C = A * B$.

Распараллелить блочную программу умножения двух матриц $C = A * B$ с использованием технологии OpenMP: перемножение каждого двух блоков выполнить параллельно.

Провести численные эксперименты и построить таблицу сравнений времени выполнения различных программных реализаций решения задачи. Определить лучшие реализации. Проверить корректность (правильность) программ.

1.2 Метод решения

Числовая матрица A размеров $m \times n$, разделенная горизонтальными и вертикальными линиями на блоки (клетки), которые представляют собой матрицы, называется блочной (клеточной) матрицей. Элементами блочной матрицы A являются матрицы A_{ij} размеров $m_i \times n_j$, $i=1,2,\dots,p$, $j=1,2,\dots,q$, причем $m_1+m_2+\dots+m_p=m$ и $n_1+n_2+\dots+n_q=n$.

Рассмотрим теперь операцию умножения блочных матриц A и B .

Блочные матрицы A и B называются согласованными, если разбиение матрицы $A=(A_{ik})$ на блоки по столбцам совпадает с разбиением матрицы $B=(B_{kj})$ по строкам, т.е. блоки A_{ik} имеют размеры $m_i \times p_k$, а блоки B_{kj} — $p_k \times n_j$ ($k=1,2,\dots,s$). У согласованных блочных матриц блоки A_{ik} и B_{kj} являются согласованными матрицами.

Произведением $C=A \cdot B$ согласованных блочных матриц **A** и **B** называется блочная матрица $C=(C_{ij})$, блоки которой вычисляются по следующей формуле:

$$C_{ij}=A_{i1} \cdot B_{1j}+A_{i2} \cdot B_{2j}+...+A_{is} \cdot B_{sj}$$

Это означает, что блочные матрицы, разделенные на блоки надлежащим образом, можно перемножать обычным способом. Чтобы получить блок C_{ij} произведения, надо выделить *i*-ю строку блоков матрицы **A** и *j*-й столбец блоков матрицы **B**. Затем найти сумму попарных произведений соответствующих блоков: первый блок *i*-й строки блоков умножается на первый блок *j*-го столбца блоков, второй блок *i*-й строки блоков умножается на второй блок *j*-го столбца и т.д., а результаты умножений складываются.

Глава 2

Хранение матриц

2.1 Матрица A

Матрица A симметричная, хранится как верхне-треугольная. Хранится в виде одномерного массива по блочным столбцам.

Размер массива, в котором хранится матрица A определяется по формуле арифметической прогрессии:

$$S_n = (n + 1) * n / 2$$

где *n* – размер матрицы A.

Рассмотрим функцию получения элемента матрицы **a_{ij}** из массива, в котором хранится матрица:

```

int Aij(int ii, int jj) {
    int k = 0; //индекс для того, чтобы сделать сдвиг, для печати нижнего треугольника
    int *index = new int[n]; //массив индексов
    for (int i = 0; i < n; ++i)
        index[i] = sumProgression(i);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            if (i > j) { //нижний треугольник
                index[j] += k;
                if (i == ii && j == jj) {
                    return A[index[j]];
                }
            }
            else { // верхний треугольник
                if (i == 0) {
                    if (i == ii && j == jj) {
                        return A[index[j]];
                    }
                }
                else {
                    index[j] += 1;
                    if (i == ii && j == jj) {
                        return A[index[j]];
                    }
                }
            }
        }
        k = i + 1;
    }
}

```

В процессе реализации задачи было замечено, что индексы каждой последующей строки для нижнего (симметричного) треугольника матрицы отличаются от индексов предыдущей строки на номер текущей строки. Аналогично, для верхнего треугольника индексы отличаются на 1. Поэтому определяется к какому треугольнику принадлежит искомый элемент и делается соответствующий сдвиг. Когда искомый элемент найден, делается return.

2.2 Матрица В

Матрица В ниже-треугольная. Хранится в виде одномерного массива по блочным строкам.

Размер массива, в котором хранится матрица В определяется по формуле арифметической прогрессии:

$$S_n = (n + 1) * n / 2$$

где n – размер матрицы B.

Рассмотрим функцию получения элемента матрицы **bij** из массива, в котором хранится матрица:

```
int Bij(int ii, int jj) {
    int k = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            if (i >= j) { //нижний треугольн
                if (i == ii && j == jj)
                    return B[k];
                ++k;
            }
            else // верхний треугольник
                if (i == ii && j == jj)
                    return 0;
    }
}
```

Т.к. матрица хранится по строкам, проблемы с индексами нет. Для получения нужного элемента, определяем к какому треугольнику он принадлежит и делаем return: искомый элемент, если он принадлежит нижнему треугольнику и 0, если принадлежит верхнему.

Глава 3

Сравнение результатов

Размеры матриц	Размер блока	Время работы поточного алгоритма	Время работы простого алгоритма
512	8	0,135653	0,481374
512	16	0,126299	0,459267
512	32	0,142882	0,598728
512	64	0,149907	0,570311

Табл. 1. – Тестирование результатов работы программы

Поточный алгоритм – потоки разделяются по блочным строкам/столбцам. Было выяснено, что такое распределение по потокам работает быстрее всего.

Как видно из таблицы поточный алгоритм работает почти в пять раз быстрее, кроме того, можно заметить, что чем меньше размер блока – тем быстрее работа программы. Исходя из этого можно сделать предположение о том, что процессы становятся в очередь, длиной из четырех, т.к. такое количество позволяет процессор и оптимальным числом блоков является

Тестирование выполнялось на процессоре Intel® Core™ i5-3570 CPU @ 3.40GHz 3.80 GHz, с кол-вом ядер равному 4.

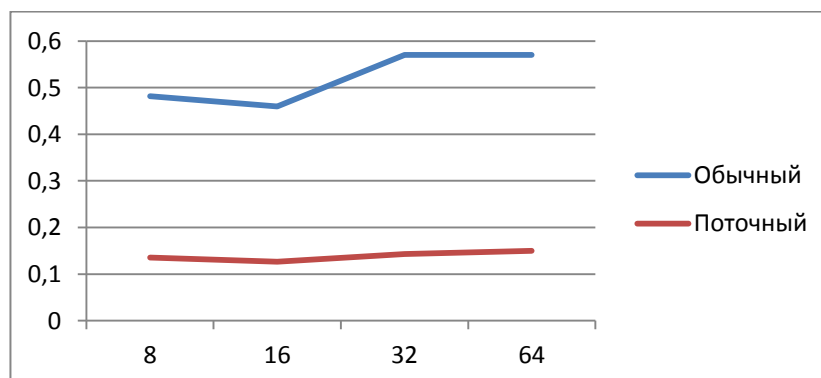


График 1. – Тестирование результатов работы программы

Размеры матриц	Размер блока	Время работы поточного алгоритма	Время работы простого алгоритма
10	2	0.00143646	0.00121204
50	2	0.536472	0.99317
100	2	13.6977	25.5146
10	5	0.000968503	0.00080282
50	5	0.542934	1.18357
100	5	14.161	26.1945
10	10	0.0011913	0.00160887
50	10	0.620508	1.05429
100	10	14.1258	25.6654

Табл.2. – Результаты работы старой программы

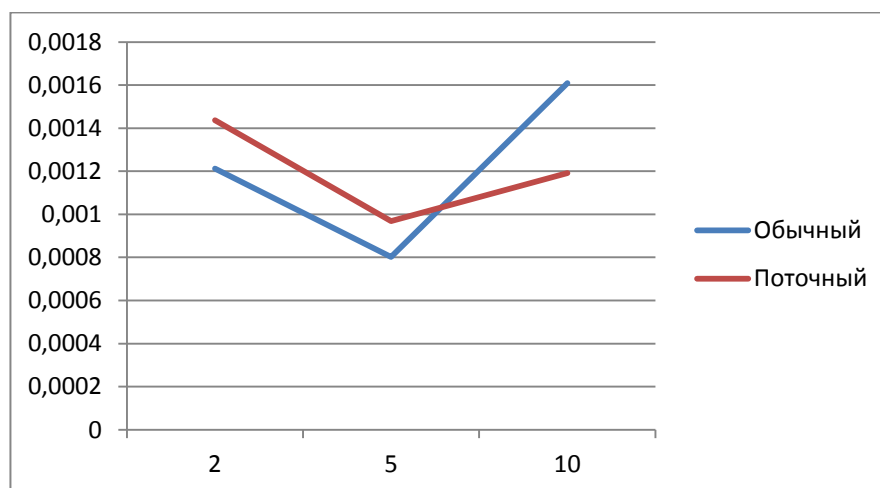


График 2. – Тестирование результатов работы программы для матрицы 10x10

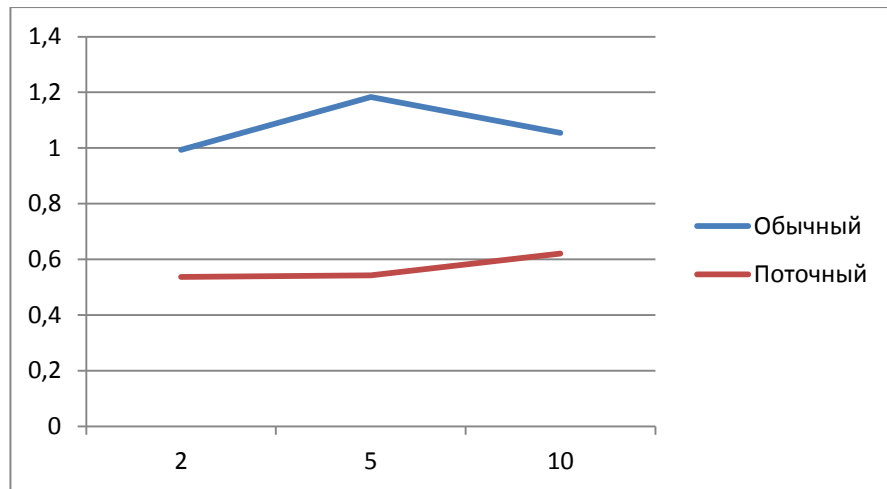


График 3. – Тестирование результатов работы программы для матрицы
50x50

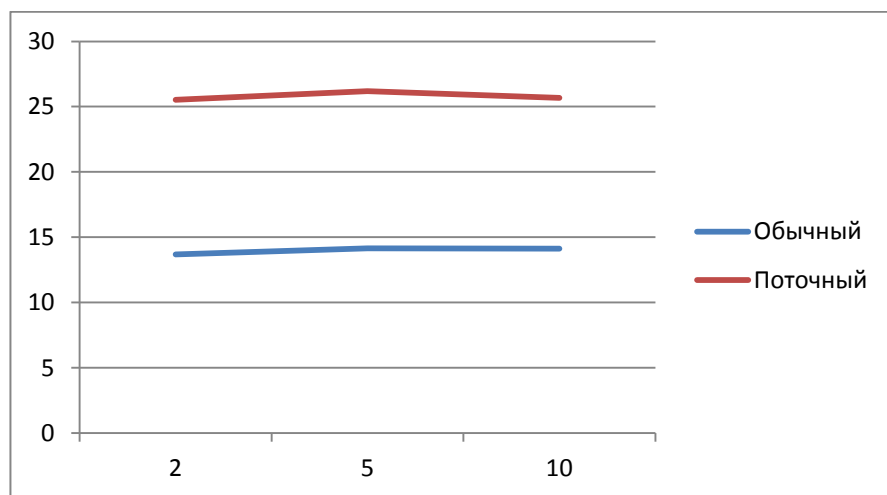


График 4. – Тестирование результатов работы программы для матрицы
100x100

Матрица А (ее индексы):

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 5 & 7 & 8 \\ 3 & 6 & 8 & 9 \end{pmatrix}$$

(по заданию) инициализируется как одномерный массив в виде:

$$(0, 1, 4, 2, 5, 7, 3, 6, 8, 9)$$

Но т.к. получение элемента из массива (видно из сравнения результатов старой программы и новой) занимает очень много времени, то сначала получаем матрицу, а затем проводим вычисление. Так возможно точно оценить время перемножения, без добавления времени вычисления элемента матрицы из массива.