МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)
Институт информационных технологий, математики и механики

Отчет по лабораторной работе
на тему:

«Умножение плотных матриц. Элементы типа double. Блочная схема, алгоритм Фокса»

Выполнил(а): студент(ка) группы 381506-2
Ухина А.А.

Нижний Новгород
2018

1 Постановка задачи

Умножение матриц — одна из основных операций над матрицами.

Пусть даны две прямоугольные матрицы *A* и *B* размерности $l \times m$ и $m \times n$ соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}.$$

Тогда матрица *C* размерностью $l \times n$:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix},$$

в которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = 1, 2, \ldots l;\ j = 1, 2, \ldots n).$$

называется их *произведением*.

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором.

В рамках данной лабораторной работы мы будем рассматривать квадратные матрицы, то есть матрицы *A*, *B* и *C* размерностью $n \times n$. Умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка.

2 Метод решения

Последовательный алгоритм

Последовательный алгоритм перемножения двух плотных матриц A и B размерности (n x n) реализуется по формуле:

$$C(i,j) = \sum_{k=0}^{n-1} A(i,k) * B(k,j), 0 \le i,j < n$$

При представлении матриц одномерным массивом получаем следующую функцию:

```
void MatrMatrMult(double * A, double * B, double * C, int N)
{
	for (int i = 0; i < N; i++)
		for (int j = 0; j < N; j++)
		{
			C[i * N + j] = 0.0;
			for (int k = 0; k < N; k++)
				C[i * N + j] += A[i * N + k] * B[k * N + j];
		}
}
```

Сложность такого алгоритма – $O(n^3)$.

Параллельный алгоритм

При построении параллельных способов выполнения матричного умножения широко используется блочное представление матриц. В этом случае результирующая матрица и матрицы-аргументы матричного умножения разделяются между потоками параллельной программы на квадратные блоки.

При таком способе разделения данных исходные матрицы A, B и результирующая матрица C представляются в виде наборов блоков. Количество блоков по горизонтали и вертикали являются одинаковыми и равными $q = \sqrt{p}$ (где p – количество процессов). Размер блоков k определяется как $k = n/q$ (где n – количество строк/столбцов в исходных матрицах).

При таком представлении данных операция умножения матриц A и B будет выглядеть следующим образом:

$$\begin{pmatrix} A_{00} & A_{01} & \dots & A_{0q-1} \\ \dots & \dots & \dots & \dots \\ A_{q-10} & A_{q-11} & \dots & A_{q-1q-1} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & \dots & B_{0q-1} \\ \dots & \dots & \dots & \dots \\ B_{q-10} & B_{q-11} & \dots & B_{q-1q-1} \end{pmatrix} =$$

$$= \begin{pmatrix} C_{00} & C_{01} & \dots & C_{0q-1} \\ \dots & \dots & \dots & \dots \\ C_{q-10} & C_{q-11} & \dots & C_{q-1q-1} \end{pmatrix},$$

Где каждый блок результирующей матрицы определяется как:

$$C_{ij} = \sum_{s=0}^{q-1} A_{is} B_{sj} \ .$$

Одним из алгоритмов, выполняющих блочное умножение матриц, является алгоритм Фокса.

3 Схема распараллеливания

Алгоритм Фокса – блочный алгоритм умножения матриц. Для корректной работы необходимо, чтобы число процессов p было полным квадратом.

За основу параллельных вычислений для матричного умножения при блочном разделении данных принят подход, при котором базовые подзадачи отвечают за вычисления отдельных блоков матрицы C и при этом в подзадачах на каждой итерации расчетов располагается только по одному блоку исходных матриц A и B. Номер подзадачи в дальнейшем мы будем обозначать (i,j), что соответствует вычисляемому блоку матрицы C.

Описание алгоритма:

1. *Этап инициализации*. Каждой подзадаче (i,j) передаются блоки A_{ij}, B_{ij} и обнуляются блоки C_{ij} на всех подзадачах;
2. *Этап вычислений*. На каждой итерации I, ($I = \overline{0, q-1}$) осуществляются следующие операции:
   a) Для каждой строки i ($i = \overline{0, q-1}$) блок A_{ij} подзадачи (i,j) пересылается на все подзадачи той же строки i решетки; индекс j, определяющий положение подзадачи в строке, вычисляется в соответствии с выражением j=(i+ I)mod q, где mod есть операция получения остатка от целочисленного деления;
   b) Полученные в результаты пересылок блоки A'_{ij}, B'_{ij} каждой подзадачи (i,j) перемножаются и прибавляются к блоку C_{ij}.
   c) Блоки B'_{ij} каждой подзадачи (i,j) пересылаются подзадачам, являющимся соседями сверху в столбцах решетки подзадач (блоки подзадач из первой строки решетки пересылаются подзадачам последней строки решетки).

Оценка эффективности для алгоритма Фокса.

Алгоритм Фокса требует для своего выполнения q итераций, в ходе которых каждый процессор перемножает свои текущие блоки матриц A и B и прибавляет результаты умножения к текущему значению блока матрицы C.

При применении последовательного алгоритма перемножения матриц число шагов имеет порядок $O(n^3)$. При применении параллельного алгоритма количество выполняемых операций будет иметь порядок $n^3 \ / \ p$. Как результат, показатели ускорения имеют вид:

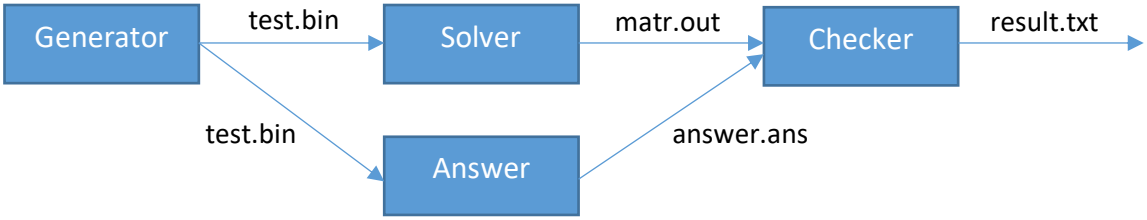
$$S_p = \frac{n^3}{(n^3/p)} = p$$

То есть теоретическое ускорение равно количеству потоков.

Для OpenMP и TBB алгоритм несколько упрощается. При выполнении параллельных алгоритмов на системах с общей памятью передача данных между процессорами уже не требуется. Поэтому алгоритм сводится к тому, что каждый процесс вычисляет свой блок результирующей матрицы C.

4 Описание программной реализации

Для проведения эксперимента была разработана следующая структура:



Программа **Generator**. Генерирует случайные матрицы заданных размеров. Полученные матрицы записывает в бинарный файл test.bin.

Программа **Solver**. Считывает матрицы из файла test.bin. Содержит реализацию параллельного алгоритма. Полученный ответ записывает в бинарный файл matr.out.

Программа **Answer**. Представляет собой реализацию последовательной версии умножения матриц. Полученный ответ считается правильным и сравнивается в дальнейшем с ответом, полученным в результате параллельного алгоритма. Ответ записывается в файл answer.ans.

Программа **Checker**. Представляет собой программу, проверяющую ответ программы Solver с правильным ответом. Результат записывает в файл result.txt.

Реализация параллельного алгоритма для *OpenMP*, для каждого потока:

1. Получить номер текущего потока ProcRank с помощью функции `omp_get_thread_num()`
2. Получить номер строки и столбца блока, соответствующего потока.
3. Вычислить элементы блока результирующей матрицы C, просуммировав необходимые произведения блоков матриц A и B.

Реализация параллельного алгоритма для *TBB*:

1. Инициализировать объект класса `task_scheduler_init init(ProcNum)`, где `ProcNum` – количество потоков.
2. Задать размер порции вычислений grainsize. Так как каждый поток вычисляет свой блок результирующей матрицы C, то переменная grainsize примет значение 1.
3. Задать итерационное пространство `blocked_range<int>(0, ProcNum, grainsize)`
4. Реализовать функтор (класс специального вида, основная функциональность которого сосредоточена в методе operator()).
    a) Функтор не должен содержать в себе ни обрабатываемые данные, ни получаемый результат, поэтому все поля являются указателями на внешние данные. Инициализация полей происходит с помощью конструктора. Для вычисления блока элементов необходимо знать матрицу A, B, размерность матриц N, размер блока, количество блоков; результат будет записан в матрицу C, поэтому их необходимо указать в конструкторе.
    b) Основной метод operator() принимает в качестве параметра итерационное пространство и производит вычисления соответствующего блока.

5. Вызвать функцию void parallel_for(const Range& range, const Body& body) (где range – итерационное пространство, body – функтор) для распараллеливания цикла.

5 Подтверждение корректности

Для проверки результата параллельного алгоритма используется программа **Checker**. Программа считывает ответ программы **Solver** и правильный ответ (полученный программой **Answer**). По полученным данным вычисляется погрешность, как сумма квадратов разности решений, и сравнивается с малой величиной 10^{-6}, если погрешность превышает данную величину, то результат считается неверным.

Checker использует класс для сохранения результата. Класс содержит в себе функции для записи в файл результата, который представляется собой вердикт, вместе с его обозначением:

- AC = Accepted – решение выдаёт корректный результат на данном тесте
- WA = Wrong Answer – решение выдаёт некорректный результат на данном тесте

время выполнения последовательного и параллельного алгоритма и ускорение, если оно имеется.

6 Результаты экспериментов

Время работы алгоритма при различном количестве узлов и размерах матриц (для *OpenMP*)

Размер матриц	**Время работы**			
	1 узел	4 узла	9 узлов	16 узлов
900	20.703	3.074	2.701	2.168
1000	26.803	4.119	3.631	2.823
1200	49.894	10.361	6.123	4.962
1400	80.778	22.243	12.365	9.369

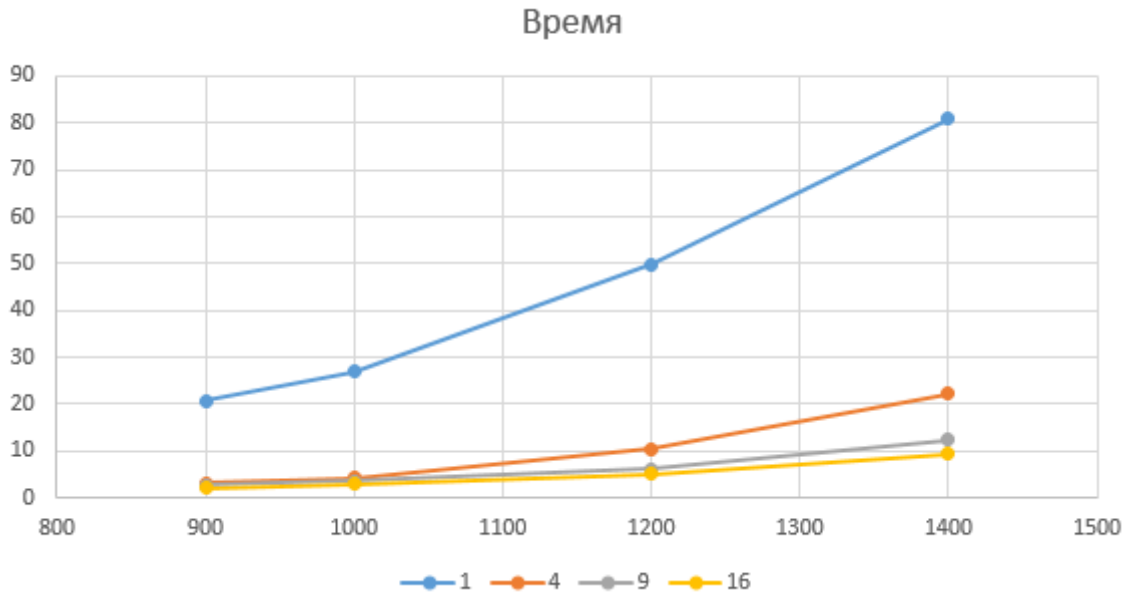


Рис.1. График зависимости времени от размера матриц

Ускорение (для *OpenMP*)

Размер матриц	**Ускорение**		
	4 узла	9 узлов	16 узлов
900	6.734	7.664	9.549
1000	6.507	7.381	9.494
1200	4.816	8.148	10.055
1400	4.362	6.532	8.621

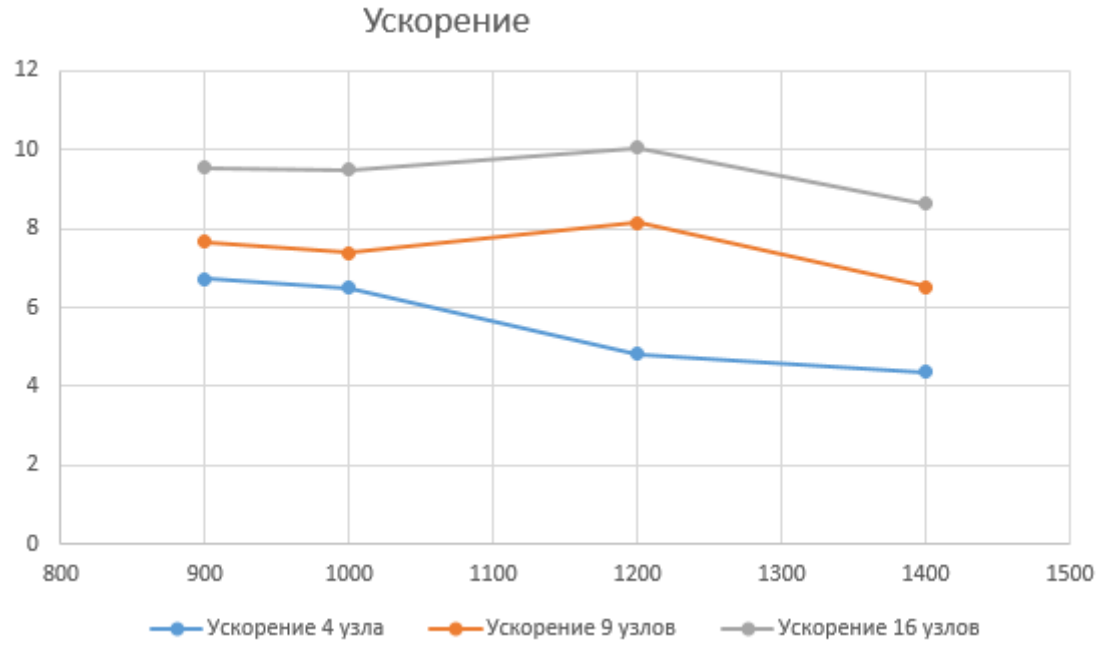


Рис.2. График зависимости ускорения от размера матриц

Время работы алгоритма при различном количестве узлов и размерах матриц (для *TBB*)

Размер матриц	**Время работы**			
	1 узел	4 узла	9 узлов	16 узлов
900	21.363	3.144	3.048	2.846
1000	26.701	4.608	4.014	3.105
1200	51.693	11.439	7.684	6.024
1400	80.524	22.243	13.617	10.109

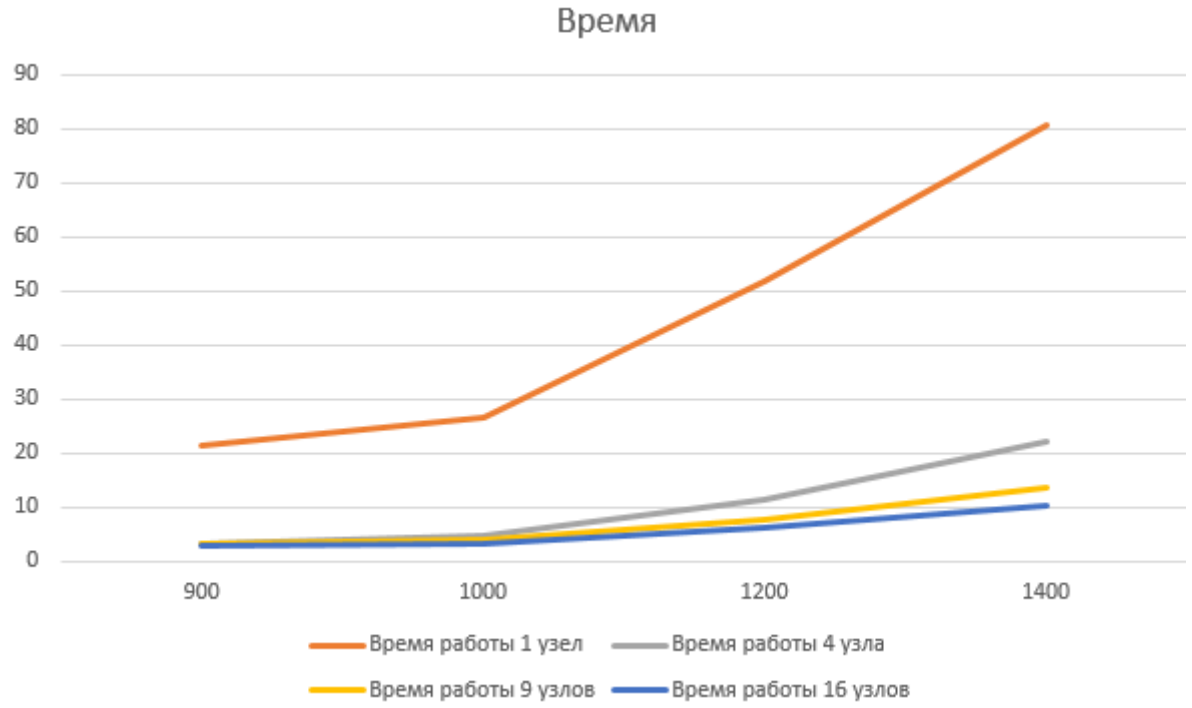


Рис.3. График зависимости времени от размера матриц

Ускорение (для *ТВВ*)

Размер матриц	**Ускорение**		
	4 узла	9 узлов	16 узлов
900	6.794	7	7.506
1000	5.794	6.65	8.59
1200	4.519	6.727	8.581
1400	3.62	5.913	7.965

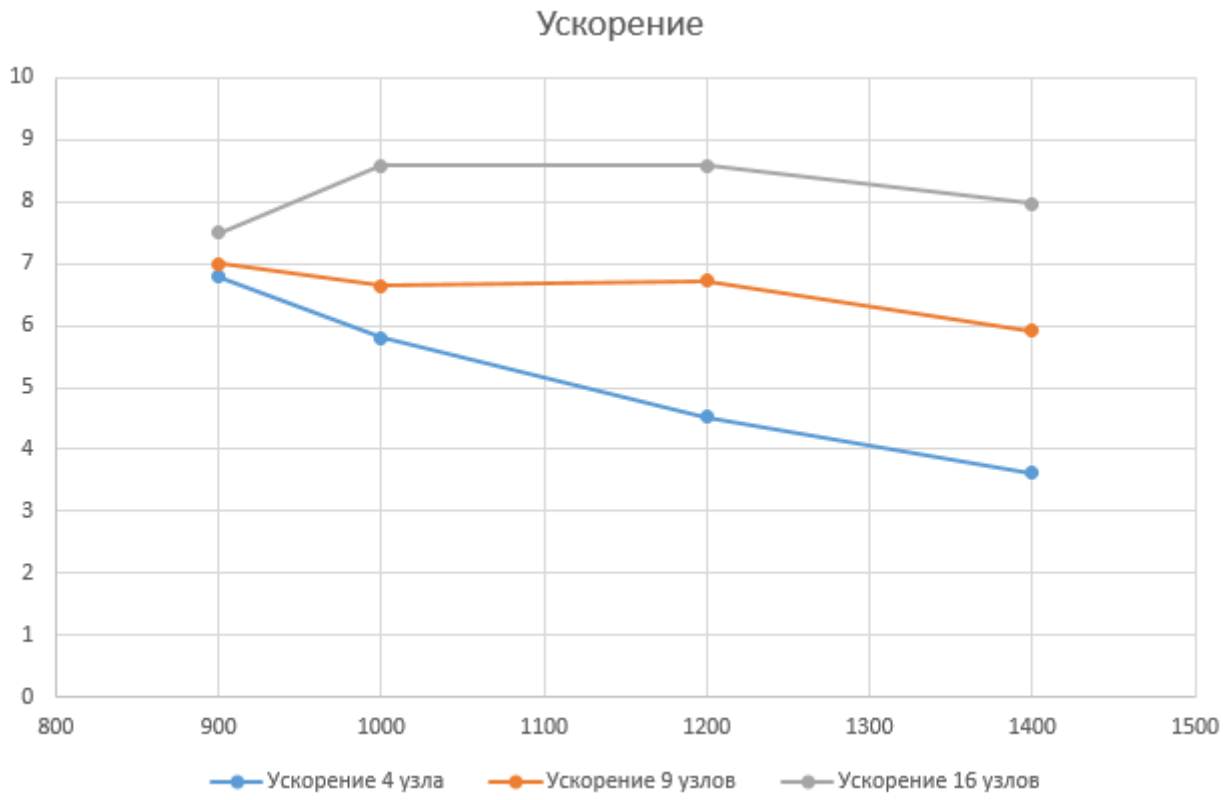


Рис.4. График зависимости ускорения от размера матриц

7 Результат

Результатом лабораторной работы является программа, выполняющая умножение квадратных матриц.

Из экспериментов видно, что в результате увеличения количества потоков время работы программы значительно уменьшается. Уже на 4 узлах наблюдается заметная разница.

Время работы OpenMP-версии и TBB-версии практически совпадают.

Полученное ускорение соответствует теоретическому. Однако на 16 потоках замечено лишь небольшое увеличение ускорения. Таким образом для получения хороших результатов достаточно использовать 4 или 9 потоков.