Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт Информационных Технологий, Математики и Механики

Отчёт к лабораторной работе

по курсу Параллельное программирование

Умножение плотных матриц.

Блочная схема, алгоритм Кэннона.

Выполнил:

студент института ИТММ гр. 381406-1

Доронин М.А.

Проверил:

доцент кафедры МОСТ, к.т.н.

Сысоев А.В.

Нижний Новгород

2017 г.

Содержание

[Введение 3](#_Toc482959823)

[Постановка задачи 4](#_Toc482959824)

[Последовательный алгоритм 5](#_Toc482959825)

[Параллельный алгоритм 5](#_Toc482959826)

[Метод Кэннона 6](#_Toc482959827)

[Анализ эффективности 7](#_Toc482959828)

[Разработка схемы параллельных вычислений 8](#_Toc482959829)

[Реализация OpenMP 8](#_Toc482959830)

[Реализация TBB 8](#_Toc482959831)

[Реализация MPI + OpenMP 8](#_Toc482959832)

[Проведение вычислительных экспериментов 9](#_Toc482959833)

[Реализация OpenMP 9](#_Toc482959834)

[Реализация MPI 10](#_Toc482959835)

[Реализация MPI + OpenMP 12](#_Toc482959836)

[Выводы 15](#_Toc482959837)

[Заключение 16](#_Toc482959838)

[Литература 17](#_Toc482959839)

[Приложения 18](#_Toc482959840)

# Введение

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Чтобы получить произведение матриц *A* и *B* необходимо, чтобы количество столбцов матрицы *A* было равно количеству строк матрицы *B*. Если условие выполняется, произведение матриц определено. Результатом умножения матрицы *A* *[n \* m]* и матрицы *B* [*m \* k]* будет матрица *C* размерностью *[n \* k]*. Каждый элемент результирующей матрицы получается умножением строки *i* матрицы *A* на столбец *j* матрицы *B*.

Операция умножения является трудоемкой операцией, и к тому же часто используемой в прикладных задачах, поэтому оптимизация ее вычисления и распараллеливание алгоритма представляет большой интерес. Для выполнения операции умножения существуют множество алгоритмов, многие из которых имеют оптимальный ресурс для распараллеливания.

Одним из семейств таких алгоритмов является блочное умножение матриц. В частности, алгоритм Кэннона, который рассмотрен в данной лабораторной работе. Идея алгоритма состоит в изменении схемы начального распределения блоков перемножаемых матриц между процессорами вычислительной системы. Начальное расположение блоков в алгоритме Кэннона подбирается таким образом, чтобы располагаемые блоки на процессорах могли бы быть перемножены без каких-либо дополнительных передач данных между процессорами. При этом подобное распределение блоков может быть организовано таким образом, что перемещение блоков между процессорами в ходе вычислений может осуществляться с использованием простых коммуникационных операций.

# Постановка задачи

Целью данной лабораторной работы является разработка программы, в которой реализован алгоритм блочного умножения плотных матриц методом Кэннона с применением принципов параллельного программирования.

Выполнение задачи включает:

1. Освоение темы (постановка задачи).
2. Изучение метода решения.
3. Разработку схемы параллельных вычислений.
4. Реализацию программы (с использованием технологий OpenMP, TBB, MPI).
5. Проведение вычислительных экспериментов (на кластере) с анализом масштабируемости.
6. Подготовку отчета с анализом результатов экспериментов.

Реализация программы должна включать последовательную и параллельную версию алгоритмов, а также сравнение получаемых результатов каждой из версий. Программа должна корректно работать на задачах малой размерности. Необходимо реализовать генерацию исходных данных по вводимым пользователем размерности матрицы, также количество используемых процессов и потоков на каждом из процессов должны задаваться пользователем. Требуется вывести время работы программы на параллельной и последовательной части и посчитать ускорение. Любые некорректные данные должны быть обработаны.

Программа должна использовать интерфейс MPI для эффективной работы на системах с распределенной памятью, и технологии OpenMP и TBB для работы с общей памятью.Метод решения

Пусть и- вещественные матрицы, для которых необходимо найти их произведение . Следует уточнить, что классический алгоритм Кэннона предполагает, что исходные матрицы являются квадратными матрицами *.* Обе матрицы разбиваются на блоков, то есть на по вертикали и по горизонтали. Отсюда следует, что для корректной работы алгоритма размерность матрицы должна быть кратна количеству блоков . Тогда размер каждого блока .

## Последовательный алгоритм

Последовательный алгоритм умножения двух плотных квадратных вещественных матриц осуществляется по формуле:

Этот алгоритм предполагает выполнение операций умножения и столько же операций сложения элементов исходных матриц, то есть верхняя асимптотическая оценка алгоритма имеет порядок O(n3). Известны последовательные алгоритмы умножения матриц, обладающие меньшей вычислительной сложностью (например, алгоритм Страссена), но эти алгоритмы сложны в освоении и реализации, поэтому в данной работе в качестве последовательных частей (то есть для умножения блоков) будет выбран именно этот алгоритм.

Однако, глобально последовательный алгоритм будет реализован методом Кэннона, но каждый из блоков будет умножаться на единственном процессе по очереди.

## Параллельный алгоритм

При работе с общей памятью, каждому потоку определяется пара чисел . Соответственно, данный поток перемножает все блоки матрицы по горизонтали и блоки матрицы по вертикали, результаты суммирует в блоке матрицы , определяющийся парой чисел .

При работе с распределенной памятью процессы выстраиваются в квадратную решетку. Задача каждого процесса – вычислить соответствующий блок результирующей матрицы. Для перемножения блоков исходных матриц может использоваться тот же последовательный алгоритм перемножения матриц, либо параллельный по принципу работы с общей памятью (в случае совмещения технологий MPI + OpenMP).

## Метод Кэннона

Метод Кэннона отличает от других алгоритмов блочного умножения начальная инициализация расположения блоков. Для простоты рассмотрим пример для матрицы размерностью .

Исходные матрицы:



Рисунок 1 - Исходный вид матриц.

Начальная инициализация блоков матриц включает выполнение следующих операций передач данных:

1. Для каждой строки  решетки подзадач блоки матрицы  сдвигаются   
   на  позиций влево;
2. Для каждого столбца  решетки подзадач блоки матрицы  сдвигаются на  позиций вверх.



Рисунок 2 - Начальная инициализация расположения блоков.

Работа алгоритма предполагает выполнение следующих операций:

1. Умножение блока и ;
2. Аккумулирование результата в блоке ;
3. Циклический сдвиг блоков матрицы А влево на одну позицию;
4. Циклический сдвиг блоков матрицы В вверх на одну позицию.



Рисунок 3 - Расположение блоков на каждой итерации.

## Анализ эффективности

Каждый из процессов, работая параллельно, умножает блоков размером , таким образом трудоемкость алгоритма Кэннона:

Тогда показатели ускорения и эффективности выражаются следующим образом:

Однако, данные показатели являются идеальными, и на практике редко достижимы по причине затраты времени на коммуникацию между процессами и потоками, на переключение контекстов и по причине наличия не распараллеливаемых частей.

# Разработка схемы параллельных вычислений

## Реализация OpenMP

1. Каждый поток получает информацию о размере матрицы , количестве блоков и номере итерации .
2. Каждый поток, зная свой номер и номер итерации, вычисляет начальные строку и столбец и конечные строку и столбец . Таким образом определяется блок матрицы – подзадача одного потока на данной итерации.
3. В «приватную» память потока копируются блоки и перемножаются тривиальным методом умножения матриц.
4. Результат аккумулируется в соответствующем блоке матрицы .



Рисунок 4 - Определение подзадачи для потока.

## Реализация TBB

Реализация алгоритма с использованием технологии TBB похожа на реализацию OpenMP с той лишь разницей, что для каждого потока создается , который инициализируется размером матриц , количеством блоков , и номером блока . Каждый поток производит блочное умножение раз и возвращает результат.

## Реализация MPI + OpenMP

1. Процессы выстраиваются в квадратную решетку с замыканием краев (решетка-тор). Процесс с координатами объявляется корневым.
2. Корневой процесс выделяет память под матрицы и и заполняет их случайными числами.
3. Каждый процесс выделяет память под блок и принимает данные от корневого процесса.
4. Каждый процесс выполняет умножение блоков в соответствии с реализацией OpenMP.
5. Все процессы передают блок матрицы влево по решетке, а блок матрицы – вверх, и принимают соответствующие блоки от других процессов посредством операции MPI\_Sendrecv\_replace.
6. После всех итераций каждый процесс передает результат корневому процессу.

# Проведение вычислительных экспериментов

В ходе данной работы были проведены результаты экспериментов с использованием технологий OpenMP, TBB и MPI. Ниже в таблицах представлены результаты:

1. Реализации OpenMP с разной размерностью матриц и разным количеством потоков.
2. Реализация MPI с разной размерностью матриц и разным количеством процессов. Без использования технологий TBB и OpenMP.
3. Реализация MPI + OpenMP с разным количеством процессов, потоков и разными размерностями матриц.

## Реализация OpenMP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | Количество потоков | | | | | |
| 1 | 4 | 9 | 16 | 25 | 36 |
| 300 | 0,0284 | 0,0127 | 0,0122 | 0,0108 | 0,0114 | 0,0124 |
| 600 | 0,411 | 0,1726 | 0,1081 | 0,0989 | 0,0879 | 0,0831 |
| 900 | 1,7487 | 1,0367 | 0,5345 | 0,3622 | 0,3274 | 0,3225 |
| 1200 | 12,8577 | 2,6324 | 2,1859 | 1,2786 | 0,9274 | 0,7164 |
| 1500 | 26,1779 | 11,752 | 5,0036 | 5,3439 | 2,49612 | 1,7773 |
| 1800 | 48,5856 | 21,2947 | 8,7324 | 8,259 | 5,978 | 4,3065 |
| 2400 | 125,056 | 57,0627 | 49,2146 | 21,3582 | 18,9089 | 16,3747 |
| 3000 | 285,718 | 122,385 | 100,822 | 96,9995 | 42,6172 | 34,4316 |

Таблица 1 - Реализация OpenMP. Время работы.

Рисунок 5 - Реализация OpenMP. График времени работы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | Количество потоков | | | | |
| 4 | 9 | 16 | 25 | 36 |
| 300 | 2,24 | 2,33 | 2,63 | 2,49 | 2,29 |
| 600 | 2,38 | 3,8 | 4,16 | 4,68 | 4,95 |
| 900 | 1,69 | 3,27 | 4,83 | 5,34 | 5,42 |
| 1200 | 4,88 | 5,88 | 10,06 | 13,86 | 17,95 |
| 1500 | 2,23 | 5,23 | 4,9 | 10,49 | 14,73 |
| 1800 | 2,28 | 5,56 | 5,88 | 8,13 | 11,28 |
| 2400 | 2,19 | 2,54 | 5,86 | 6,61 | 7,64 |
| 3000 | 2,33 | 2,83 | 2,95 | 6,7 | 8,3 |

Таблица 2 - Реализация OpenMP. Ускорение.

Рисунок 6 - Реализация OpenMP. График ускорения.

## Реализация MPI

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | Количество процессов | | | | | |
| 1 | 4 | 9 | 16 | 25 | 36 |
| 1440 | 22,5915 | 5,6313 | 2,5428 | 0,9113 | 1,0191 | 1,1382 |
| 2160 | 80,1972 | 22,0291 | 9,43887 | 5,7151 | 3,487 | 2,1457 |
| 2880 | 395,094 | 58,2825 | 28,0483 | 13,6624 | 10,5208 | 6,5266 |
| 3600 | 438,343 | 109,444 | 45,9853 | 27,2597 | 18,2219 | 13,3323 |
| 4320 | 843,115 | 197,168 | 92,0712 | 48,609 | 32,395 | 22,995 |

Таблица 3 - Реализация MPI. Время работы.

Рисунок 7 - Реализация MPI. График времени работы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | Количество потоков | | | | |
| 4 | 9 | 16 | 25 | 36 |
| 1440 | 4,01 | 8,88 | 24,79 | 22,17 | 19,85 |
| 2160 | 3,64 | 8,5 | 14,03 | 23 | 37,38 |
| 2880 | 6,78 | 14,09 | 28,92 | 37,55 | 60,54 |
| 3600 | 4,01 | 9,53 | 16,08 | 24,06 | 32,88 |
| 4320 | 4,28 | 9,16 | 17,34 | 26,03 | 36,67 |

Таблица 4 - Реализация MPI. Ускорение.

Рисунок 8 - Реализация MPI. График ускорения.

## Реализация MPI + OpenMP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | Количество потоков | | | | | |
| 1 | 4 | 9 | 16 | 25 | 36 |
| 1440 | 5,6313 | 1,4717 | 1,3846 | 1,1737 | 1,2296 | 1,2584 |
| 2160 | 22,0291 | 16,8704 | 4,5147 | 3,758 | 4,0055 | 3,6928 |
| 2880 | 58,2825 | 43,8719 | 32,2348 | 10,4826 | 17,3513 | 11,0817 |
| 3600 | 109,444 | 95,5315 | 79,5489 | 58,5084 | 20,6068 | 17,7348 |
| 4320 | 197,168 | 172,36 | 147,1 | 129,151 | 92,4914 | 36,6271 |

Таблица 5 - Реализация MPI + OpenMP. Время работы. Количество используемых процессов: 4.

Рисунок 9 - Реализация MPI + OpenMP. График времени работы. Количество используемых процессов: 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | Количество потоков | | | | |
| 4 | 9 | 16 | 25 | 36 |
| 1440 | 3,83 | 4,07 | 4,8 | 4,58 | 4,47 |
| 2160 | 1,31 | 4,88 | 5,86 | 5,5 | 5,97 |
| 2880 | 1,33 | 1,81 | 5,56 | 3,36 | 5,26 |
| 3600 | 1,15 | 1,38 | 1,87 | 5,31 | 6,17 |
| 4320 | 1,14 | 1,34 | 1,53 | 2,13 | 5,38 |

Таблица 6 - Реализация MPI + OpenMP. Ускорение, относительно конфигурации: 4 процесса + 1 поток. Количество используемых процессов: 4.

Рисунок 10 - Реализация MPI + OpenMP. График ускорения, относительно конфигурации: 4 процесса + 1 поток. Количество используемых процессов: 4.

Рисунок 11 - Реализация MPI + OpenMP. График ускорения, относительно конфигурации: 1 процесс + 1 поток. Количество используемых процессов: 4.

Рисунок 12 - Реализация MPI + OpenMP. График ускорения, относительно конфигурации: 1 процесс + 1 поток. Количество используемых процессов: 36.

# Выводы

По результатам проведенных экспериментов можно сделать следующие выводы:

Алгоритм Кэннона на больших размерностях матриц, в среднем, показывает ускорение, равное числу участвующих процессов, то есть близко к линейному (что соответствует теоретическим оценкам сложности алгоритма).

Алгоритм Кэннона может работать как быстрее, так и медленнее теоретической оценки. На размере матриц 2880\*2880 и конфигурации 36 процессов + 1 поток было достигнуто суперлинейное ускорение. Это явление можно объяснить удачным расположением блоков матриц в кэше процессора, что ускоряет работу с памятью при работе отдельно взятого процесса. Для выполнения последовательного алгоритма необходимы матрицы в полном объеме, а их полностью в кэше разместить не удается, в результате чего большое количество времени тратится на обращение к оперативной памяти.

На небольших размерностях матриц Алгоритм Кэннона показывает ускорение ниже теоретического, так как затраты на коммуникации между процессами нивелирует выигрыш от параллельных вычислений. Также добавление большого числа потоков при большом количестве процессов может снижать ускорение, так как результирующие подзадачи имеют слишком малую размерность и затраты на коммуникации превышают время работы.

Результаты экспериментов на кластере хорошо демонстрируют приведенные выводы. При большом количестве доступных процессов матрицы делятся на небольшие, по отношению к исходному размеру матриц, блоки, которые могут быть закешированы процессором, в результате чего операции над ними выполняются быстро.

Так же стоит отметить, что OpenMP и TBB работают с небольшой разностью во времени как в ту, так и в другую сторону.

# Заключение

В ходе лабораторной работы был реализован алгоритм Кэннона умножения плотных матриц. Реализация программы включает последовательную и параллельную часть с использованием технологий OpenMP, TBB и MPI. Проведен теоретический анализ эффективности, проведены эксперименты по времени работы последовательной и параллельной части, на основании которых сделаны выводы о практических оценках эффективности. Программа была проверена на данных малой размерности и на некорректных данных.

Эксперименты проводились на домашнем ПК и на кластере ННГУ.

# Литература

1. В.В.Воеводин, Ю.А.Кузнецов. Матрицы и вычисления. М.: Наука, 1984.
2. Воеводин В.В. Вычислительные основы линейной алгебры. М.: Наука, 1977.

# Ссылки

1. Репозиторий, содержащий разработанную программную реализацию [https://github.com/Maxim-Doronin/CannonMethod].