

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Параллельные алгоритмы»
Тема: Параллельное умножение матриц

Студент гр. 0303

Бодунов П.А.

Преподаватель

Сергеева Е.И.

Санкт-Петербург

2023

Цель работы.

Исследовать масштабируемость умножения матриц и алгоритм Штрассена.

Задание.

4.1 Реализовать параллельный алгоритм умножения матриц с масштабируемым разбиением по потокам.

Исследовать масштабируемость выполненной реализации с реализацией из работы 1.

4.2 Реализовать параллельный алгоритм “быстрого” умножения матриц (Штрассена или его модификации).

Проверить, что результаты вычислений реализаций 4.1 и 4.2 совпадают.

Сравнить производительность с реализацией 4.1 на больших размерностях данных (порядка 10^4 – 10^6)

Выполнение работы.

Создание матрицы происходит с помощью функции:

`void generate_matrix(int rows, int columns)` – на вход подается количество строк и столбцов. Элементы матрицы заполняются случайными целочисленными значениями в диапазоне $[0, 99]$.

Создание матриц:

`void generate_matrices(vector<vector<int>>& matrix1, vector<vector<int>>& matrix2)` – на вход подаются ссылки на 2 матрицы, которые необходимо сгенерировать.

Умножение матриц (лаб. 1) для одного потока происходит при помощи функции:

`void thread_multiply_matrices(vector<vector<int>>& matrix1,
vector<vector<int>>& matrix2, vector<vector<int>>& matrix_res, int start, int end)` – на вход подаются ссылки на матрицы, которые необходимо перемножить, ссылка на матрицу результата перемножения, индексы `start` и `end` – индексы строк, которые перемножает поток.

Умножение матриц (лаб. 1) происходит при помощи функции:

`vector<vector<int>> lab1_mult(vector<vector<int>>& matrix1,
vector<vector<int>>& matrix2, int num_threads)` – на вход подаются ссылки на матрицы, которые необходимо перемножить и количество потоков, которые будут заниматься перемножением матриц. Функция возвращает результирующую матрицу.

Параллельный алгоритм умножения матриц с масштабируемым разбиением по потокам для одного потока происходит при помощи функции:

`void thread_scalable_mult_matrix(vector<vector<int>>& matrix1,
vector<vector<int>>& matrix2, vector<vector<int>>& matrix_res, int threads_num,
int threads_i)` – на вход принимает ссылки на матрицы, которые необходимо перемножить, ссылка на матрицу результата перемножения, количество потоков, индекс текущего потока.

Параллельный алгоритм умножения матриц с масштабируемым разбиением по потокам происходит при помощи функции:

`vector<vector<int>> scalable_mult(vector<vector<int>>& matrix1,
vector<vector<int>>& matrix2, int threads_num)` – на вход принимает ссылки на матрицы, которые необходимо перемножить, ссылка на матрицу результата перемножения, количество потоков. Функция возвращает результирующую матрицу.

Взятие квадратного среза матрицы осуществляется при помощи функции:

`vector<vector<int>> half_slice(vector<vector<int>>& matrix, int y, int x, int size)` – на вход подается матрица, координаты начала среза y и x , размер среза. Функция возвращает срез матрицы.

Расширение матрицы до размеров 2^k происходит при помощи функции:
`void extension_matrix(vector<vector<int>>& matrix, int dim)` – на вход подается ссылка на матрицу и размерность до которой расширяется матрица.

Сложение или вычитание 2-х матриц:
`vector<vector<int>> sum_matrices(vector<vector<int>> matrix1, vector<vector<int>> matrix2, char oper)` – на вход подается матрицы и оператор сложения или вычитания. Функция возвращает результирующую матрицу.

Запись части матрицы в результирующую:
`void write_res(vector<vector<int>>& res, vector<vector<int>> matrix, int y, int x, int size)` – на вход подается ссылка на результирующую матрицу, матрицу размерности меньше, которую необходимо переписать в результирующую, а так же координаты и размер результирующей матрицы.

Рекурсивная функция Алгоритма Штрассена:
`void strassen_mult_matrix(vector<vector<int>> matrix1, vector<vector<int>> matrix2, vector<vector<int>>& matrix_res, int cur_rec_depth, int allow_rec_depth)` – на вход подаются 2 матрицы, которые необходимо перемножить, ссылка на результирующую матрицу, глубина рекурсии и максимальная глубина рекурсии.

Функция подготавливающая исходные матрицы к применению Алгоритма Штрассена, а так же запуск Алгоритма Штрассена:

`vector<vector<int>> strassen_mult(vector<vector<int>>& matrix1, vector<vector<int>>& matrix2, int threads_num)` – на вход подаются ссылки на

матрицы, которые необходимо перемножить, количество потоков. Функция возвращает результирующую матрицу.

Функция сравнения 2-х матриц:

`bool compare_matrices(vector<vector<int>>& matrix1, vector<vector<int>>& matrix2)` – на вход подаются ссылки на матрицы, которые необходимо сравнить. Функция возвращает `true`, если матрицы равны, иначе `false`.

Исследование скорости работы перемножения матриц алгоритма из лаб. 1, алгоритмом с масштабируемым умножением, алгоритма Штрассена

Для этого возьмём постоянное количество потоков равное 7.

Результаты зависимости времени работы программы от размерности матрицы для алгоритма из лаб. 1, алгоритмом с масштабируемым умножением, алгоритма Штрассена представлены в табл. 1.

Таблица 1 — Зависимость времени работы программы от размерности матрицы

Размерность матрицы	Время выполнения в мкс.		
	Алгоритм лаб. 1	Алгоритм с масштабируемым умножением	Алгоритм Штрассена
32	242	331	1868
64	1683	1390	4630
128	7461	5549	11921
256	59121	40141	64981
512	480437	325861	386106
1024	4210226	2981161	2567880
1025	4179962	3080455	20291854
2048	34690027	26814837	18207876
4096	343486874	263039504	149555069

Исходя из результатов таблицы 1, время выполнения на маленьких матрицах меньше для алгоритма из лаб.1, потому что не все потоки в данном алгоритме равно нагружены, время выполнения на больших матрицах меньше для алгоритма Штрассена, т. к. асимптотика времени выполнения у данного алгоритма меньше. Но если взять размер матрицы не кратный степени двойки, то алгоритм Штрассена работает дольше, чем алгоритм с масштабируемым умножением.

Выводы.

В процессе выполнения лабораторной работы были изучены и практически реализованы алгоритм масштабируемого умножения матриц и алгоритм Штрассена, а так же практически доказано, что алгоритм Штрассена работает быстрее для больших размеров матриц кратных степени двойки.