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. 0381 _____ Соколов Д. В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2022

Цель работы.

Реализовать параллельный алгоритм умножения матриц. Исследовать масштабируемость выполненной реализации. Реализовать параллельный алгоритм «быстрого» умножения матриц Штрассена.

Задание.

4.1 Реализовать параллельный алгоритм умножения матриц. Исследовать масштабируемость выполненной реализации.

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.

Для выполнения параллельного умножения матриц классы *Matrix* и *MatrixHandler* были дополнены функциями реализующими умножение для конкретного элемента в результирующей матрице (*partial_mult()* в *Matrix*), а также функциями умножающими некоторое количество элементов в заданном отрезке (*multiply_part()*) и непосредственно функции параллельного умножения двух матирц.

Функционал, необходимый для работы алгоритма Штрассена был вынесен в отдельный класс Strassen. Были написаны функции для приведения матрицы к размерам, совместимым с алгоритмом (ближайшая степень 2), разбиение и сборки матриц из частей и непосредственно рекурсивная функция *Strassen_algorithm()*, выполняющая умножение двух матриц алгоритмом Штрассена.

Алгоритм:

1. Проверить совместимость размеров матриц, если размеры не поддерживаются алгоритмом – дополнить матрицы нулями так, чтобы получилась квадратная матрица размером ближайшей степени 2.
2. Проверить не достигла ли матрица предельно маленького размера для того, чтобы ее можно было умножить в лоб (если да, то возвращаем результат обычного последовательного умножения).
3. Проверить не вышла ли глубина рекурсивного вызова за заданный предел. (если вышла, то следующий шаг выполняем последовательно)
4. Запустить параллельный алгоритм Штрассена.
   4.1. Разбить матрицы A и B на 4 подматрицы каждую (a и b, соответственно) размера n/2, где n – размер исходной матрицы.
   4.2. В **отдельных потоках** вызвать расчет 7 p матриц, являющихся перемножением линейных комбинаций a и b матриц.
   4.3. Дождаться **при помощи std::promise** значений матриц p и получить значения подматриц c результирующей матрицы C при помощи линейных комбинаций матриц p.
   4.4. Собрать результирующую матрицу C и вернуть ее.
5. Если же условия в пунктах 3. и 4. не выполнены, значит будет вызван последовательный алгоритм Штрассена, логика действий аналогична пунткам 4.1 – 4.4, но **без использования отдельных потоков и std::promise**.

Ниже представлены конкретные линейные комбинации, использующиеся в алгоритме:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

Рис. 1 – Деление исходных и результирующей матриц.

$$\begin{aligned}
\mathbf{P}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{P}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{P}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{P}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{P}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{P}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{P}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

Рис. 2 – Вычисление матриц p.

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{P}_1 + \mathbf{P}_4 - \mathbf{P}_5 + \mathbf{P}_7 \\
\mathbf{C}_{1,2} &= \mathbf{P}_3 + \mathbf{P}_5 \\
\mathbf{C}_{2,1} &= \mathbf{P}_2 + \mathbf{P}_4 \\
\mathbf{C}_{2,2} &= \mathbf{P}_1 - \mathbf{P}_2 + \mathbf{P}_3 + \mathbf{P}_6
\end{aligned}$$

Рис 3. – Вычисление матриц c.

Для программы также было реализовано thread-safe логирование, с использованием стэка Трайбера из предыдущей лабораторной работы, в классе L□gger. L□gger имеет несколько уровней логирования, в программе используются только 2: *INFO* – информационные сообщения, *TRACE* – отслеживание работы алгоритма. При помощи функции *toggle_log()* можно задать уровень логирования в программе, по умолчанию – *INFO*.

Полная работа программы заключается в генерации 2 случайных матриц, с заданными размерами. Перемножению их сначала параллельным умножением Matri□□andler, затем параллельным умножением Strassen, замером времени выполнения обеих операций, вывода результатов в файл и проведения валидации результатов (сравнения полученных матриц на равенство) при помощи перегруженного оператора равенства для класса Matri□. Также на всех ключевых шагах исполнения программы осуществляется логирование.

Пример работы программы:

```
Matrix dimensions: [16x16],
filled with integer numbers with [10] digits.
Simple parallelization: matrix multiplication split between [4] threads;
Strassen algorithm runs in parallel until reaching [3] OR
until matrices reach [4]
[INFO] Simple parallelization finished within: 1079 mcs
[INFO] Strassen finished within: 10267 mcs
[INFO] Validation: 1

Process finished with exit code 0
```

В таблице 1 представлено сравнение времени работы алгоритма в зависимости от размера входных данных. Количество потоков для параллельного умножения – 7. Максимальная глубина рекурсии для алгоритма Штрассена – 5, предельный минимальный размер – 64□64.

Размер данных	Время работы алгоритма умножения, мс	Время работы алгоритма Штрассена, мс
128□128	54	36
256□256	277	174
512□512	1491	825
1024□1024	11071	5655

2048□2048	126937	52463

Таблица 1. – Эксперимент с различными размерами матриц.

Реализованный алгоритм Штрассена значительно выигрывает по времени на больших размерах матриц, но проигрывает на маленьких матрицах. Это обусловлено тем, что создание потоков – дорогостоящая относительно производительности операция, а в параллельном исполнении алгоритма Штрассена количество потока растет по экспоненциальному закону.

Выводы.

В ходе выполнения лабораторной работы была написана программа на языке программирования C++ для параллельного умножения матриц. Также был реализован алгоритм Штрассена.