МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий математики и механики**

**ОТЧЕТ К ЛАБОРАТОРНОЙ РАБОТЕ**

**«Реализация метода Штрассена для умножения матриц»**

**Выполнил**:

студент группы 381506-1

Сунцов С.И.

**Проверил:**

Кустикова В.Д.

             Нижний Новгород

Оглавление

[Постановка задачи. 3](#_Toc514609854)

[Метод решения. 3](#_Toc514609855)

[Описание последовательной реализации. 4](#_Toc514609856)

[Описание схемы распараллеливания на OpenMP и TBB. 4](#_Toc514609857)

[Описание параллельного алгоритма на TBB. 5](#_Toc514609858)

[Описание программной реализации. 5](#_Toc514609859)

[Подтверждение корректности. 6](#_Toc514609860)

[Результаты экспериментов по оценке масштабируемости. 6](#_Toc514609861)

[Вывод 9](#_Toc514609862)

## Постановка задачи.

Требуется разработать программу, которая умножает две матрицы методом Штрассена. Программа должна состоять из следующих частей:   
 1. Последовательный алгоритм  
 2. Параллельный алгоритм с использованием библиотеки OpenMP.

3. Параллельный алгоритм с использованием библиотеки TBB (Threading Building Blocks ).

Также в программе должны быть реализованы следующие подпрограммы:  
1. Generator-генерация тестовых матриц заданного размера.  
2. Checker-проверка корректности полученных результатов умножения.  
3. Typer-ввод данных (исходных матриц) для программы из txt файла.

4. Viwer-сохранение результата умножения в txt файл.

## Метод решения.

Рассмотрим алгоритм Штрассена, который умножает плотные квадратные матриц размера *N × N*, где . Если размер матриц не является степенью двойки, то мы можем дополнить исходные матрицы столбцами и строками из нулей до нужного размера.

Умножение двух матриц записывается как , где *A*, *B*, *C* матрицы размера *N × N.*

Разделим матрицы *A*, *B* и *C* на равные по размеру блочные матрицы:

,где

Тогда имеем следующие формулы вычисления блоков матрицы C:

Однако с помощью этой процедуры нам не удалось уменьшить количество умножений. Как и в обычном методе, нам требуется 8 умножений.

Теперь определим новые элементы (формулы Штрассена):

Таким образом, нам нужно всего 7 умножений на каждом этапе рекурсии. Теперь выразим :

Рекурсивный процесс продолжается, до тех пор, пока размер матриц *C*i,j не станет достаточно малым, далее используют обычный метод умножения матриц. Это делают из-за того, что алгоритм Штрассена теряет эффективность по сравнению с обычным на малых матрицах в силу большего числа сложений. Оптимальный размер матриц для перехода к обычному умножению зависит от характеристик процессора и языка программирования и на практике лежит в пределах от 32 до 128. Эта числовая характеристика называется порогом.

## Описание последовательной реализации.

После того, как были получены исходные матрицы и их размер, требуется проверить размер матриц, равен ли он степени двойки. Если размер не равен , то требуется проинициализировать три матрицы размером , где это число максимально близкое к размеру исходных матриц. Далее проверяем размер матрицы с заранее выбранным порогом, если значении меньше, то умножаем матрицы стандартным алгоритмом умножения двух матриц, иначе мы разбиваем исходные матрицы на 4 подматрицы размером . После разбиения рекурсивно вычисляем значения   
Вычисляем блоки результирующей матрицы и формируем из них матрицу с результатом умножения. Если размер матрицы не был равен степени 2, то убираем «лишние»

нулевые столбцы и строчки.

## Описание схемы распараллеливания на OpenMP и TBB.

Основная идея распараллеливания алгоритма Штрассена с помощью OpenMP и TBB заключается в разделении 7 рекурсивных умножений между потоками. При заданном количестве потоков в ходе алгоритма рекурсивные умножения будут распределяться поочередно между разными потоками.   
  
Описание параллельного алгоритма на OpenMP.

Проверяется размер матриц, полученных на вход алгоритма:

* если размер меньше или равен заданному порогу, то выполняется стандартный алгоритм умножения двух матриц.
* иначе:
  + размер матриц уменьшается в два раза.
  + исходные матрицы разбиваются на 4 блока и записываются в соответствующие вспомогательные матрицы.
  + 7 рекурсивных умножений по формулам Штрассена поочередно разделяются между каждым потоком (с помощью директивы #pragma omp sections или #pragma omp task), и каждый поток выполняет своё рекурсивное умножение.
  + вычисляются блоки результирующей матрицы.
  + результирующая матрица формируется из полученных блоков.

Описание параллельного алгоритма на TBB.

Для реализации параллельного алгоритма на TBB требуется создать специальный класс **StrassenTask**.

1. Выполняется создание и запуск главной задачи **StrassenTask**

2. Проверяется размер матриц, пришедших на вход алгоритма:

* если размер меньше или равен заданному порогу, то выполняется стандартный алгоритм умножения двух матриц.
* иначе:
  + размер матриц уменьшается в два раза.
  + исходные матрицы разбиваются на 4 блока и записываются в соответствующие вспомогательные матрицы.
  + создаётся список подзадач для главной задачи **StrassenTask**.
  + 7 рекурсивных умножений по формулам Штрассена разделяются между каждым потоком. Каждый поток создает подзадачу **StrassenTask** (с помощью new(this.allocate\_child()), при этом список подзадач увеличивается на единицу.
  + происходит добавление списка подзадач в очередь готовых к выполнению и идёт ожидание выполнения всех подзадач (с помощью spawn\_and\_wait\_for\_all()).
  + вычисляются блоки результирующей матрицы.
  + результирующая матрица формируется из полученных блоков.

## Описание программной реализации.

* Lab3 - решение задачи последовательным алгоритмом и создание бинарного файла с результатом.
* generator - программа для генерации набора тестовых данных и их сохранение в бинарные файлы.

Через командную строку пользователь вводит номер теста, который соответствует индексу в массиве. Значения массива – числа, равные количеству элементов массива. Далее в бинарный файл записывается следующее: размер матриц, сгенерированные матрицы.

* checker - программа для проверки корректности параллельных версий.
* OmpV- решение задачи параллельным алгоритмом с помощью OpenMP и sections и создадние бинарного файла с результатом.
* OmpV2- решение задачи параллельным алгоритмом с помощью OpenMP и task и создадние бинарного файла с результатом.
* TBB- решение задачи параллельным алгоритмом с помощью TBB и создадние бинарного файла с результатом.
* viwer- считывание данных из бинарного файла в текстовый.
* typer - считывание данных из текстового файла в бинарный.
* Набор тестов (папка tests) для проверки корректности параллельных версий.

## Подтверждение корректности.

Для подтверждения корректности следует запустить программу checker, которая сравнит результаты умножения матриц методом Штрасена и стандартным умножением. Результат сравнения будет записан в файл result.txt.

## Результаты экспериментов по оценке масштабируемости.

Таблица 1 Время выполнения OpenMP версии на основе sections

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | | 900x900 | 1024x1024 | 1500x1500 | 2048x2048 |
| Время выполнения последовательной версии | | 0.0034 | 2.3670 | 8.1567 | 16.5636 |
| Время выполнения параллельной версии OMP на основе sections | 2 потока | 0.0032 | 2.2847 | 7.9395 | 16.4567 |
| 4 потока | 0.0029 | 2.1510 | 7.8965 | 16.2064 |
| 8 потоков | 0.0025 | 2.0067 | 7.7435 | 16.1006 |
| 16 потоков | 0.0020 | 1.9001 | 7.6433 | 15.8003 |

Рисунок 1. Зависимость времени выполнения от числа потоков

Таблица 2 Ускорение в OpenMP версии на основе sections

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число потоков /Размер матрицы | 900x900 | 1024x1024 | 1500x1500 | 2048x2048 |
| 2 потока | 1.0625 | 1.0360 | 1.0273 | 1.0064 |
| 4 потока | 1.1724 | 1.1004 | 1.0329 | 1.022 |
| 8 потоков | 1.36 | 1.1795 | 1.0533 | 1.0287 |
| 16 потоков | 1.7 | 1.2457 | 1.0671 | 1.048 |

Рисунок 2 Ускорение OpenMP версии

Таблица 3 Время выполнения OpenMP версии на основе task

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | | 900x900 | 1024x1024 | 1500x1500 | 2048x2048 |
| Время выполнения последовательной версии | | 0.0034 | 2.3670 | 8.1567 | 16.5636 |
| Время выполнения параллельной версии OMP на основе task | 2 потока | 0.0027 | 1.9564 | 6.2353 | 15.5235 |
| 4 потока | 0.0026 | 1.4546 | 6.1245 | 15.5223 |
| 8 потоков | 0.002 | 1.3535 | 5.9451 | 15.2144 |
| 16 потоков | 0.0015 | 1.2355 | 5.6 | 14.990 |

Рисунок 3. Зависимость времени выполнения от числа потоков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число потоков /Размер матрицы | 900x900 | 1024x1024 | 1500x1500 | 2048x2048 |
| 2 потока | 1,259259259 | 1,209875281 | 1,308148766 | 1,067001643 |
| 4 потока | 1,038461538 | 1,344974563 | 1,018091273 | 1,000077308 |
| 8 потоков | 1,3 | 1,074695235 | 1,030176111 | 1,020237407 |
| 16 потоков | 1,333333333 | 1,095507892 | 1,061625 | 1,01496998 |

Таблица 4 Ускорение в OpenMP версии на основе task

Рисунок 4 Ускорение OpenMP версии на основе task

Таблица 5 Время выполнения TBB

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | | 900x900 | 1024x1024 | 1500x1500 | 2048x2048 |
| Время выполнения последовательной версии | | 0.0034 | 2.3670 | 8.1567 | 16.5636 |
| Время выполнения параллельной версии TBB | 2 потока | 0.0026 | 2.0538 | 6.1234 | 13.5235 |
| 4 потока | 0.0021 | 1.9423 | 5,659 | 13.5223 |
| 8 потоков | 0.0015 | 0.9671 | 5.4064 | 13.4631 |
| 16 потоков | 0.0007 | 0.9955 | 4.9432 | 13.2546 |

Рисунок 5. Зависимость времени выполнения от числа потоков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число потоков /Размер матрицы | 900x900 | 1024x1024 | 1500x1500 | 2048x2048 |
| 2 потока | 1,307692308 | 1,152497809 | 1,332054088 | 1,224801272 |
| 4 потока | 1,238095238 | 1,057406168 | 1,082063969 | 1,000088742 |
| 8 потоков | 1,4 | 2,008375556 | 1,046722403 | 1,004397204 |
| 16 потоков | 2,142857143 | 0,971471622 | 1,093704483 | 1,015730388 |

Таблица 6 Ускорение в TBB

Рисунок 6 Ускорение TBB

## Вывод

В результате выполнения лабораторной работы был реализован алгоритм Штрассена с применением таких библиотек распараллеливания, как TBB и OpenMP.

Было выяснено, что для получения эффективного времени работы программы требуется использовать различные методы и библиотеки для распараллеливания алгоритма и тестировать их время работы. В качестве примера использования различных механизмов можно считать распараллеливание с помощью task и srctions в OpenMP. Основывая на полученные данные, можно утверждать, что алгоритм Штрассена показывает ускорение близкое к линейному.