Национальный исследовательский университет “Высшая Школа Экономики”,
Факультет компьютерных наук
Департамент программной инженерии

«Программа по нахождению ранга произвольной матрицы с использованием библиотеки OpenMP»

Пояснительная записка к разработке консольного приложения

Исполнитель:
Студент группы БПИ199
Волохов Никита Алексеевич

Оглавление

1. Текст задания

Формулировка задания: «Определить ранг матрицы. Входные данные: целое положительное число n, произвольная матрица A размерности n x n. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков» [1].

2. Применяемые расчетные методы

a. Построчное заполнение произвольной матрицы

Квадратная матрица ***matrixA*** заполняется пользователем при помощи консоли (размер матрицы ***MATRIX_SIZE*** передается в качестве первого аргумента ***main***). Матрица заполняется построчно.

b. Распараллеливание программы

Как известно, ранг матрицы равен размерности наибольшего минора, детерминат которого отличен от нуля [2], соответственно, для квадратной матрицы размерности ***MATRIX_SIZE*** необходимо рассчитать ранг каждого минора размерности ***m*** на отрезке [1; ***MATRIX_SIZE***]. Задача определения ранга каждого минора размерности ***m*** была поделена на потоки (количество потоков ***THREAD_NUM*** = второму аргументу ***main***) для более эффективной работы программы. Для распараллеливания программы была использована библиотека OpenMP [3].

Все потоки запускают стартовую функцию для потоков ***func***, входной параметр которого – размерность матрицы и количество потоков для подсчета минора матрицы. Создание новых потоков в данной функции завершается, если найденный ранг минора равен размерности матрицы.

После вычисления потоком ранга минора, результат записывается в переменную ***rang*** и ***maxRang*** (в переменную ***maxRang*** посчитанный потоком ранг записываеися при условии, если посчитанный ранг больше уже лежащего в переменной значения. До всех вычислений здесь хранится 0) и в консоль выводится результат работы потока (строка формата *“Thread #<thread number> calculated rank of minor минора* ***matrixSize*** *x* ***matrixSize****:* ***rang****”*). Данный блок кода – критическая секция (если к данному блоку кода дать доступ сразу нескольким потокам, значение переменной ***maxRang*** или вывод в консоль сообщения могут быть искажены).

Операции записи посчитанных рангов в переменную ***maxRang*** и вывод результата работы потока в консоль могут конфликтовать между потоками (если, например, одна и та же операция выполняется одновременно разными потоками), поэтому их необходимо синхронизировать. Синхронизация реализована при помощи библиотеки OpenMP. Распараллеливание подсчета каждого минора матрицы при помощи директивы ***#pragma omp num_threads(n)***, а критическая секция, описанная в параграфе выше, выделена при помощи директивы ***#pragma omp critical***.

c. Вычисление ранга мартицы

Для вычисления ранга матрицы используется функция ***computeRank***, аргумент которой – один из миноров матрицы ***matrixA***. Для данного минора находится наибольший минор, детерминант которого отличен от нуля – его размерность и есть ранг минора.

3. Тестовые примеры

Программа корректно работает на корректных параметрах (см. Рисунок 1). Как можно увидеть, потоки перестают создаваться после того, как один из рангов матрицы равен ее размерности

```
$ clang++ -Xpreprocessor -fopenmp -std=c++11 -I/usr/local/include -L/usr/local/lib -lomp main.cpp -o main && ./main 5 3
Enter matrix numbers for line #1
87
71
17
54
80

Enter matrix numbers for line #2
41
90
95
22
66

Enter matrix numbers for line #3
29
75
13
98
45

Enter matrix numbers for line #4
0
9
14
20
65

Enter matrix numbers for line #5
21
84
83
47
4

Thread #3 calculated rank of minor 5 x 5: 5
Thread #2 calculated rank of minor 3 x 3: 3
Thread #1 calculated rank of minor 1 x 1: 1
Matrix A rank is 5
```

Рисунок 1. n = numOfThreads

Программа корректно работает при использовании одного потока (см. Рисунок 2)

```
$ clang++ -Xpreprocessor -fopenmp -std=c++11 -I/usr/local/include -L/usr/local/lib -lomp main.cpp -o main && ./main 3 1
Enter matrix numbers for line #1
77
78
36

Enter matrix numbers for line #2
10
41
29

Enter matrix numbers for line #3
29
99
76

Thread #1 calculated rank of minor 1 x 1: 1
Thread #1 calculated rank of minor 2 x 2: 2
Thread #1 calculated rank of minor 3 x 3: 3
Matrix A rank is 3
```

Рисунок 2. n < numOfThreads

Во всех приведенных тестах выше ранг матрицы равен ее размерности, поэтому для большей наглядности был добавлен тест, где ранг матрицы не равен ее размерности (см. Рисунок 3)

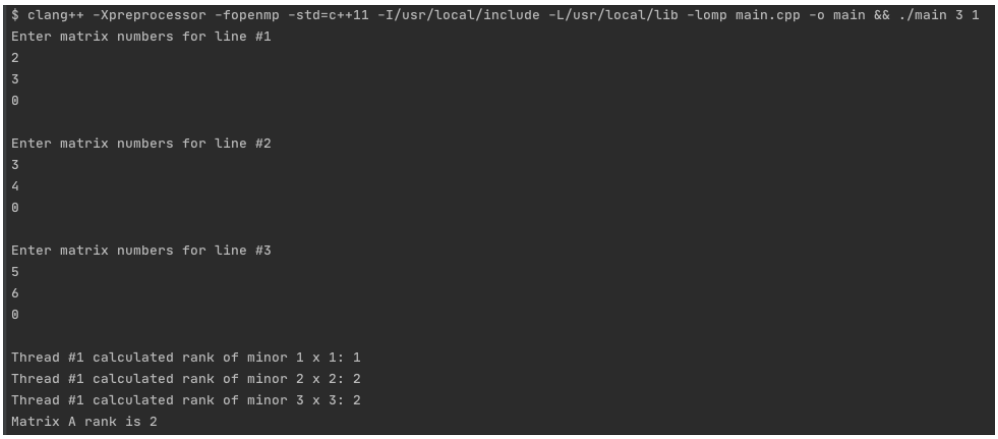

Рисунок 3. Ранг матрицы не равен ее размерности

4. Список используемых источников

[1] Практические приемы построения многопоточных приложений. [Электронный ресурс]. // URL: http://softcraft.ru/edu/comparch/tasks/t03/ (Дата обращения: 16.11.2020, режим доступа: свободный)

[2] Как найти ранг матрицы? [Электронный ресурс]. // URL: http://mathprofi.ru/rang_matricy.html (Дата обращения: 16.11.2020, режим доступа: свободный)

[3] OpenMP. [Электронный ресурс]. // URL: https://www.openmp.org/ (Дата обращения: 28.11.2020, режим доступа: свободный)

5. Текст программы

```
#include <time.h>
#include <string>
#include <vector>
#include <cmath>
#include <iostream>
#include <omp.h>

// Launch prog command (tested on unix system)
// clang++ -Xpreprocessor -fopenmp -std=c++11 -I/usr/local/include -
L/usr/local/lib -lomp main.cpp -o main && ./main <first_arg> <second_arg>

using namespace std;

// Матрица A.
vector<vector<int>> matrixA;
// Размерность матрицы A.
int n;
// Максимальный ранг подматрицы.
int maxRang = 0;

const double EPS = 1E-9;

int computeRank(vector<vector<int>> A) {
    int n = A.size();
    int m = A[0].size();

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && abs(A[j][i]) > EPS)
                break;
        }

        if (j != n) {
            ++rank;
            row_selected[j] = true;

            for (int p = i + 1; p < m; ++p)
                A[j][p] /= A[j][i];

            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i]) > EPS) {
                    for (int p = i + 1; p < m; ++p)
                        A[k][p] -= A[j][p] * A[k][i];
                }
            }
        }
    }
    return rank;
}

// Возвращает подматрицу размером m x m.
vector<vector<int>> getSubMatrix(vector<vector<int>> matrix, int m) {
    vector<vector<int>> subMatrix;

    for (int i = 0; i < m; ++i) {
        vector<int> tempVector;
```

```
            subMatrix.push_back(tempVector);

            for (int j = 0; j < m; ++j) {
                subMatrix[i].push_back(matrix[i][j]);
            }
        }

        return subMatrix;
}

// Стартовая функция для дочерних потоков.
void func(int matrixSize, int numOfThreads ) {
    #pragma omp parallel num_threads(numOfThreads)
    {
        // Квадратная подматрица размерностью m x m (m на отрезке [1; n]).
        vector<vector<int>> subMatrix;
        int rang;

        #pragma omp for
        for (int i = 1; i <= matrixSize; i++) {
            if (!(maxRang == matrixSize)) {
                if (i == n) {
                    subMatrix = matrixA;
                } else {
                    subMatrix = getSubMatrix(matrixA, i);
                }

                rang = computeRank(subMatrix);

                #pragma omp critical
                {
                    maxRang = rang > maxRang ? rang : maxRang;

                    printf("Thread #%d calculated rank of minor %d x %d: %d\n",
omp_get_thread_num() + 1, i, i, rang);
                }
            }
        }
    }
}

int main(int argc, char* argv[]) {
    const int MATRIX_SIZE = n = stoi(argv[1]);
    const int THREAD_NUM = stoi(argv[2]);
         srand(time(NULL));
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        cout << "Enter matrix numbers for line #" << i + 1 << endl;
        vector<int> tempVector;
        matrixA.push_back(tempVector);

        for (int j = 0; j < MATRIX_SIZE; ++j) {
//            string numToAdd;
//            cin >> numToAdd;
//            matrixA[i].push_back(stoi(numToAdd));
            matrixA[i].push_back(rand() % 100);
            cout << matrixA[i][j] << endl;
        }


        printf("\n");
    }
```

```
    func(MATRIX_SIZE, THREAD_NUM);

    cout << "Matrix A rank is " << maxRang << endl;
}
```