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

«Программа по нахождению ранга произвольной матрицы с использованием библиотеки POSIX Threads»

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 ***n*** 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 ***n*** необходимо рассчитать ранг каждого минора размерности ***m*** на отрезке [1; ***n***]. Задача определения ранга каждого минора размерности ***m*** была поделена на потоки (количество потоков ***numOfThreads*** = второму аргументу ***main***) для более эффективной работы программы. Для распараллеливания программы была использована библиотека POSIX Threads [3].

Существует три варианта распараллеливания программы:

- ***numOfThreads*** = 1: вычисление определителя каждого потока осуществляется при помощи одного главного потока.
- ***numOfThreads*** < ***n***: сначала запускается главный поток для вычисления ранга минора размерностью 1, затем в цикле создаются (***pthread_create***) запускаются (***pthread_join***) по ***numOfThreads*** потоков, пока не будут посчитаны ранги всех миноров.
- ***numOfThreads*** >= ***n***: сначала запускается главный поток для вычисления ранга минора размерностью 1, затем в цикле создаются (***pthread_create***) запускаются (***pthread_join***) n потоков, достаточных для вычисления ранга матрицы ***matrixA***.

Все потоки запускают стартовую функцию для потоков ***func***, входной параметр которого – размерность минора.

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 *“Поток посчитал ранг минора **m** x **m**: **rang**”*).

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 двоичного семафора ***pthread_mutex***. Описанные выше две операции ограничены ***pthread_mutex_lock*** (перед выполнением операций) и ***pthread_mutex_unlock*** (после выполнения операций), которые ограничивают доступ к данному участку кода другим потокам, если эти операции сейчас выполняются каким-либо другим потоком.

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 входных параметрах n = numOfThreads (на рисунке приведен пример n = numOfThreads = 5) (см. Рисунок 1)

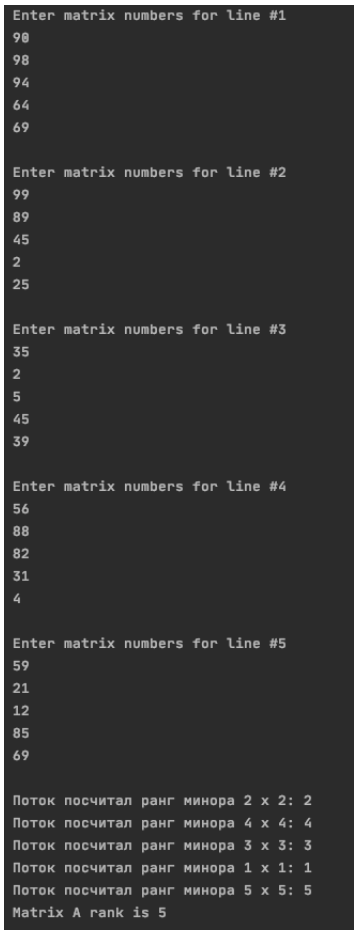

Рисунок 1. n = numOfThreads

Программа корректно работает на входных параметрах n < numOfThreads (на рисунке приведен пример n = 3, numOfThreads = 5) (см. Рисунок 2)

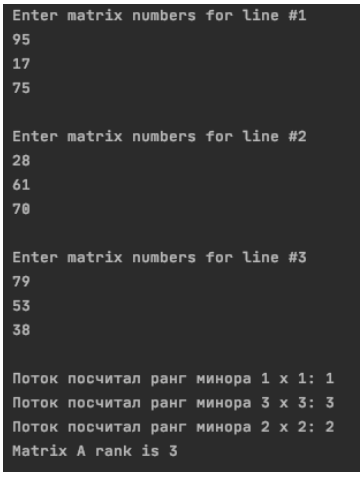

Рисунок 2. n < numOfThreads

Программа корректно работает на входных параметрах n > numOfThreads (на рисунке приведен пример n = 4, numOfThreads = 2) (см. Рисунок 3)

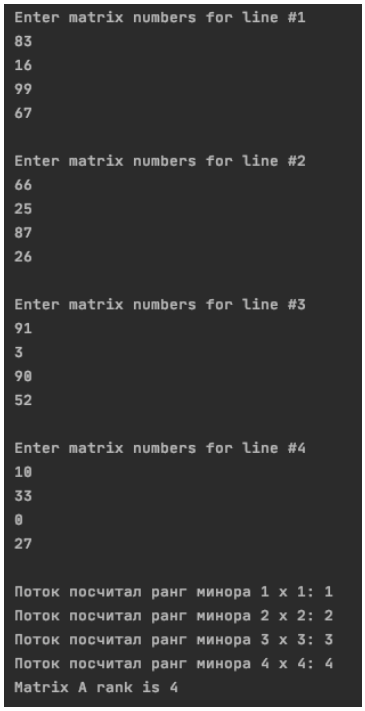

Рисунок 3. n > numOfThreads

Программа корректно работает на входных параметрах numOfThreads = 1 (на рисунке приведен пример n = 3, numOfThreads = 1) (см. Рисунок 4)

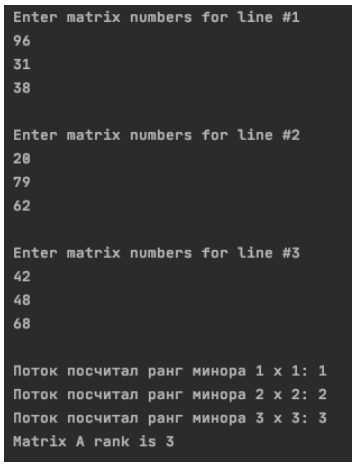

Рисунок 4. numOfThreads = 1

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 5)

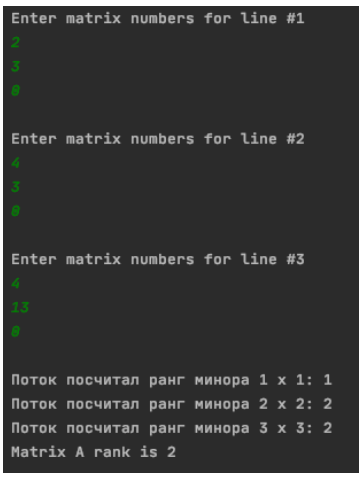

Рисунок 5. Ранг матрицы не равен ее размерности

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] POSIX Threads. [Электронный ресурс]. // URL: https://ru.wikipedia.org/wiki/POSIX_Threads (Дата обращения: 16.11.2020, режим доступа: свободный)

5. Текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string>
#include <iostream>
#include <pthread.h>
#include <vector>
#include <cmath>

using namespace std;

// Двоичный семафор.
pthread_mutex_t mutex1;
// 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(void* param) {
        // Размерность подматрицы m x m.
        int m = *(int*)param;
        // Квадратная подматрица размерностью m x m (m на отрезке [1; n]).
        vector<vector<int>> subMatrix;
        if (m == n) {
            subMatrix = matrixA;
        } else {
            subMatrix = getSubMatrix(matrixA, m);
        }

        int rang = computeRank(subMatrix);

        // Протокол входа в КС: закрыть двоичный семафор.
        pthread_mutex_lock(&mutex1);

        maxRang = rang > maxRang ? rang : maxRang;

        printf("Поток посчитал ранг минора %d x %d: %d\n", m, m, rang);

        // Протокол выхода из КС: открыть двоичный семафор.
        pthread_mutex_unlock(&mutex1);
    }

    int main (int argc, char *argv[]) {
        int n = 3;
        int numOfThreads = 5;
        matrixA;

        // Инициализация двоичного семафора
        pthread_mutex_init(&mutex1, NULL);

        srand(time(NULL));

        for (int i = 0; i < n; ++i) {
            cout << "Enter matrix numbers for line #" << i + 1 << endl;
            vector<int> tempVector;
            matrixA.push_back(tempVector);

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

            printf("\n");
        }

        // Идентификаторы для дочерних потоков.
        pthread_t threads[numOfThreads - 1];
```

```
    int num[n];

    for (int i = 0; i < n; i++) {
        // Номера размерности подматрицы для потоков.
        num[i] = i + 1;
    }

    if (numOfThreads == 1) {
        // Если поток один, то через главный находим ранг матрицы.
        int counter = 0;

        for (int i = 0; i < n; ++i) {
            func((void *)(num + counter));
            ++counter;
        }
    } else {
        if (numOfThreads < n) {
            // Нахождение ранга матрицы размерностью 1 x 1.
            func((void *)num);

            int counterCreate = 0;
            int counterJoin = 0;

            do {
                for (int i = 0; i < numOfThreads - 1; ++i) {
                    if (counterCreate == n - 1) {
                        break;
                    }

                    pthread_create(&threads[i], NULL, func, (void*)(num +
counterCreate + 1));
                    ++counterCreate;

                }

                // Нахождение ранга матрицы размерностей кроме 1 x 1.
                for (int i = 0; i < numOfThreads - 1; ++i) {
                    if (counterJoin == n - 1) {
                        break;
                    }

                    pthread_join(threads[i], NULL);
                    ++counterJoin;
                }
            } while (counterCreate < n - 1);
        } else {
            for (int i = 0; i < n - 1; ++i) {
                pthread_create(&threads[i], NULL, func, (void*)(num + i + 1));
            }
            // Нахождение ранга матрицы размерностью 1 x 1.
            func((void *)num);

            // Нахождение ранга матрицы остальных размерностей.
            for (int i = 0; i < n - 1; ++i) {
                pthread_join(threads[i], NULL);
            }
        }
    }

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