

Правительство Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования «Национальный исследовательский
университет «Высшая школа экономики»
Факультет компьютерных наук
Департамент программной инженерии

Пояснительная записка к домашнему заданию
По дисциплине
«Архитектура вычислительных систем»
3 вариант

Работу выполнил:
Студент группы БПИ-194, Аникеев Егор Васильевич
Преподаватель:
Легалов Александр Иванович

Москва 2020

Задание

Найти алгебраическое дополнение для каждого элемента матрицы. целое положительное число n , произвольная матрица A размерности $n \times n$. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

Решение

Алгебраическим дополнением элемента матрицы называется число $A_{ij} = (-1)^{i+j} M_{ij}$, где M_{ij} - дополняющий минор, определитель матрицы, полученной из матрицы A путем вычеркивания i строки и j столбца.

Так можно выделить часть, которая будет исполняться в различных потоках.

Для поиска определителя матрицы будем использовать разложение матрицы по

строке, например:
$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix};$$
 и так будем делать рекурсивно до тех пор, пока не получим матрицу размера 2×2 или 1×1 , определитель которых можно вычислить гораздо проще.

Теперь необходимо определить модель построения многопоточного вычисления. Наиболее подходящей является модель итеративного параллелизма. Основанием для этого являются следующие факторы:

- Потоки идентичны, отличием является только для какого элемента матрицы вычисляется алгебраическое дополнение;
- Внутри каждого потока существуют итерации;
- Синхронизация потока происходит благодаря общим переменным с исходной матрицей и с матрицей алгебраических дополнений.

Для решения проблемы, когда количество поток меньше количества элементов в матрице, был выбран следующий подход: каждый поток вычисляет алгебраическое дополнение для k элементов матрицы, где $k = (n * n) / f$, где n - количество строк и столбцов в матрице, f - заданное количество доступных потоков.

Для исключения возникновения критических ситуаций, таких как чтение элементов из исходной матрицы и записи в результирующую матрицу алгебраических дополнений, используется мьютекс.

Для работы с потоками были выбраны встроенные возможности C++, такие как `std::thread` и `std::mutex`. Для синхронизации поток применялись методы `mutex.lock()` и `mutex.unlock()`.

Формат входных данных

В первой строке вводится путь к файлу с исходной матрицей, элементы которой - целые числа. Во второй строке путь к файлу, в который будет записана матрица, состоящая из алгебраических дополнений к элементам исходной матрицы. В третьей строке количество строк и столбцов в матрице. В четвертой строке доступное число потоков.

Формат выходных данных

В файл, который был введен в консоль, будет записана матрица алгебраических дополнений.

Текст программы

```
#include <iostream>
#include <thread>
#include <fstream>
#include <vector>
using namespace std;

mutex matrMutex;

vector<vector<long long>> getMatrWithoutLine(vector<vector<long
long>> matr, int line, int column){
    int n = matr.size();
    vector<vector<long long>> newMatr;
    for (int i = 0; i < n; ++i) {
        if(i == line) continue;
        vector<long long> line;
        for (int j = 0; j < n; ++j) {
            if(j == column) continue;
            line.push_back(matr[i][j]);
        }
        newMatr.push_back(line);
    }
    return newMatr;
}

long long getDet(vector<vector<long long>> matr){
    if(matr.size() == 1){
        return matr[0][0];
    }
    if(matr.size() == 2){
        return matr[0][0] * matr[1][1] - matr[0][1] * matr[1][0];
    }
    int n = matr.size();
    long long det = 0;
    int sign = 1;
    for (int i = 0; i < n; ++i) {
        det += sign*getDet(getMatrWithoutLine(matr, 0, i));
        sign *= -1;
    }
    return det;
}

void getComplementElements(int i, int count, vector<vector<long
long>> matr,vector<vector<long long>> &complementMatr){
    for (int k = 0; k < count; ++k) {
        if(i / matr.size() >= matr.size()){
            return;
        }
    }
}
```

```

        vector<vector<long long>> newMatr;
        matrMutex.lock();
        newMatr = getMatrWithoutLine(matr, i / matr.size(), i %
matr.size());
        matrMutex.unlock();

        long long det = getDet(newMatr);
        det *= ((i / matr.size() + 1) + (i % matr.size() + 1)) % 2 ==
0 ? 1 : -1;
        matrMutex.lock();
        complementMatr[i / matr.size()][i % matr.size()] = det;
        matrMutex.unlock();
        i++;
    }
}

vector<vector<long long>> complementMatrix(vector<vector<long long>>
matr, int threadsCount){
    vector<thread> threads;
    vector<vector<long long>> newMatr;
    for (int i = 0; i < matr.size(); ++i) {
        vector<long long> newLine;
        for (int j = 0; j < matr.size(); ++j) {
            newLine.push_back(0);
        }
        newMatr.push_back(newLine);
    }
    int elemsInThread = ((long long)(matr.size() * matr.size()) /
threadsCount + 1);
    int i = 0;
    for (int k = 0; k < threadsCount; ++k) {
        threads.push_back(thread(getComplementElements, i,
elemsInThread, matr, ref(newMatr)));
        i = i + elemsInThread;
    }
    for (int k = 0; k < threads.size(); ++k) {
        threads[k].join();
    }
    return newMatr;
}

int main() {
    string inputFile, outputFile;
    int n;
    int threadsCount;
    cout << "Write input file name: ";
    cin >> inputFile;
    cout << "Write output file name: ";
    cin >> outputFile;

```

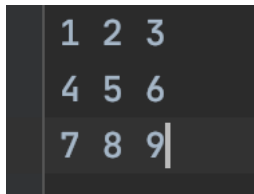
```

    cout << "Write matrix size n: ";
    cin >> n;
    cout << "Write threads count: ";
    cin >> threadsCount;
    ifstream ifs;
    ifs.open(inputFile);
    vector<vector<long long>> matr;
    for (int i = 0; i < n; ++i) {
        vector<long long> line;
        for (int j = 0; j < n; ++j) {
            int elem;
            cin >> elem;
            line.push_back(elem);
        }
        matr.push_back(line);
    }
    ifs.close();
    ofstream ofs;
    auto comp = complementMatrix(matr, threadsCount);
    for (int i = 0; i < comp.size(); ++i) {
        for (int j = 0; j < comp.size(); ++j) {
            ofs << comp[i][j] << " ";
        }
        ofs << endl;
    }
    return 0;
}

```

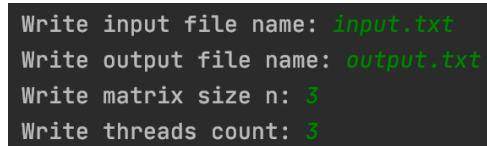
Изображения тестов

Тест 1:



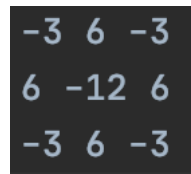
1 2 3
4 5 6
7 8 9

Данные в файле input.txt



```
Write input file name: input.txt  
Write output file name: output.txt  
Write matrix size n: 3  
Write threads count: 3
```

Данные, введенные в консоль



-3 6 -3
6 -12 6
-3 6 -3

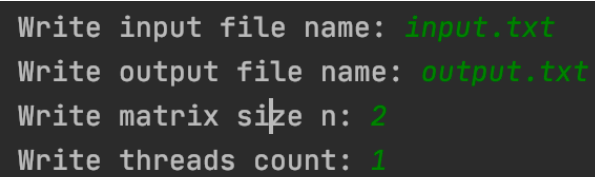
Результат работы программы в файле output.txt

Тест 2:



1 2
2 1

Данные в файле input.txt



```
Write input file name: input.txt  
Write output file name: output.txt  
Write matrix size n: 2  
Write threads count: 1
```

Данные, введенные в консоль



1 -2
-2 1

Результат работы программы в файле output.txt

Как и следовало ожидать для матрицы размера 2×2 $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \rightarrow$

$$\begin{pmatrix} a_{22} & -a_{21} \\ -a_{12} & a_{11} \end{pmatrix}.$$

Список используемых источников

1. <http://www.williamspublishing.com/PDF/5-8459-0388-2/part.pdf> - статья о способах параллельных вычислений;
2. <https://en.cppreference.com/w/cpp/thread/thread> - документация по `std::thread` в C++;
3. <http://scrutator.me/post/2012/04/04/parallel-world-p1.aspx> - статья про `thread` и `mutex` в C++.