Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Пояснительная записка заданию №4
По дисциплине

“Архитектура вычислительных систем”

Исполнитель
Студентка группы БПИ194
А.И. Гилазутдинова
Вариант 6

Москва 2020

Оглавление

1. Постановка задачи

1) Вычислить прямое произведение множеств A1, A2, A3, A4. Входные данные: множества чисел A1, A2, A3, A4, мощности множеств могут быть не равны между собой и мощность каждого множества больше или равна 1. Количество потоков является входным параметром.
2) Вывести результат алгоритма в консоль или файл.

2. Уточнение задачи

1) Реализовать программу, написанную на языке C++ 14.0, которая будет принимать на вход 4 подмножества множества целых чисел, мощности которых могут быть не равны между собой и мощность каждого множества больше или равна 1. Количество потоков является входным параметром.
2) Разработать программу с применением стандарта OpenMP и протестировать ее на нескольких примерах.
3) Формат входных и выходных данных на усмотрение разработчика.

3. Теоретическая справка

Прямое, или **декартово произведение двух множеств** — множество, элементами которого являются все возможные упорядоченные пары элементов исходных множеств. [1]

$$A_1 \times \ldots \times A_n = \left\{ \left(a_{1_{i_1}}, \ldots, a_{n_{i_n}} \right), a_{1_{i_1}} \in A_1, \ldots, a_{n_{i_n}} \in A_n \right\}$$

OpenMP (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран.[2]

Итеративный параллелизм используется для реализации нескольких процессов (часто идентичных), каждый из которых содержит циклы. Процессы программы, являющиеся итеративными программами, работают совместно над решением одной задачи; они взаимодействуют и синхронизируются либо с помощью разделяемых переменных, либо передачей сообщений. Такой параллелизм характерен, прежде всего, синхронизируемых параллельных вычислений.[3, стр. 21]

Если программа содержит большое число операций, которые могут выполняться параллельно, то такие алгоритмы обычно называют *алгоритмами с* **массовым параллелизмом**. [3, стр. 24]

4. Реализация программы

4.1.Выбор модели построения

Наиболее подходящая для этой программы модель построения многопоточных приложений – итеративная, так как потоки работают совместно над решением одной и той же задачи (нахождение прямого произведения множеств) и каждый из потоков выполняет итеративную функция в виде вычисления элементов прямого произведения множеств.

4.2. Алгоритм

Имеем 4 множества: A, B, C, D (они же A1, A2, A3, A4). Количество элементов в прямом произведении этих множеств number_of_elems = $|A| \times |B| \times |C| \times |D|$. Если number_of_threads – количество потоков, то number_of_elems ÷ number_of_threads = num_of_operations_per_thread – количество элементов, которые нужно вычислить каждому потоку[1].

То есть первый поток вычисляет элементы с 0 по num_of_operations_per_thread - 1, второй – с num_of_operations_per_thread по num_of_operations_per_thread * 2 – 1, третий – с num_of_operations_per_thread * 2 по num_of_operations_per_thread * 3 – 1 и т. д.

Следовательно будет использоваться технология распараллеливания итераций циклов, отводящая каждому потоку примерно равное количество итераций.

4.3.Реализация алгоритма

Для реализации потоков был выбран стандарт для распараллеливания программ – OpenMP с его директивой #pragma omp parallel for для распараллеливания циклов.

В нашем случае 4 цикла, и если все циклы имеют простейшую форму, можно указать компилятору, чтобы он делил на части не только итерации внешнего цикла, но итерации внутреннего, то есть все эти циклы рассматриваются как один большой цикл с парой счетчиков. Для этого нужно в директиве указать параметр collapse и за ним в скобках — число «склеиваемых» вложенных циклов, в нашем случае – 4. Условие private указывает на то, что каждый поток должен иметь свою копию итеративных переменных на всем протяжении своего исполнения.

Организация входных и выходных данных

Входные данные должны быть представлены в виде файла с данными. Результат выполнения также записывается в файл. Пользователь вводит в консоль путь к input и output файлам.

Требование к input файлу:

- В первой строке файла – элементы множества A через пробел;
- Во второй строке файла – элементы множества B через пробел;
- В третьей строке файла – элементы множества C через пробел;

[1] Кроме последнего, так как number_of_elems не всегда может быть кратно num_of_threads и тогда последний поток должен вычислить все оставшиеся элементы

- В четвертой строке файла – элементы множества D через пробел;
- В пятой строке файла число number_of_threads – количество потоков;

Данные должны быть корректны (однако есть проверка на произвольные символы в строках и количество строк в файлк).

Сам код программы предоставлен в Приложении.

5. Тестирование программы

Все тесты (входные и выходные файлы) расположены в папке **tests**. В папке **tests/input** – входные файлы (тесты), в папке **tests/output** – выходные файлы (результат работы программы), в папке **tests/console** – консоль во время того или иного теста.

6. Заключение

В результате выполнения работы был реализован алгоритм, вычисляющий прямое произведение множеств A1, A2, A3, A4 и проверенный на корректность.

7. Использованная литература

1) https://ru.wikipedia.org [Электронный ресурс]/ wikipedia.org Режим доступа: https://ru.wikipedia.org/wiki/Прямое_произведение , свободный (Дата обращения: 15.11.2020).
2) https://ru.wikipedia.org [Электронный ресурс]/ wikipedia.org Режим доступа: https://ru.wikipedia.org/wiki/OpenMP , свободный (Дата обращения: 27.11.2020).
3) Карепова Е. Д., Кузьмин Д. А., Легалов А.И., Редькин А. В., Удалова Ю. В., Федоров Г. А. Средства разработки параллельных программ. Учебное пособие. Красноярск 2007

Приложение

```
/* 6 вариант
 * Вычислить прямое произведение множеств A1, A2, A3, A4.
 * Входные данные: множества чисел A1, A2, A3, A4,
 * мощности множеств могут быть не равны между собой и мощность
 * каждого множества больше или равна 1. Количество потоков является входным
 * параметром. входные параметры: файл вида
 *
 * A1 A2 ... An
 * B1 B2 ... Bm
 * C1 C2 ... Ck
 * D1 D2 ... Dl
 * <num_of_threads>
 *
 * где
 * Ai - элементы первого множества, целые
 * Bi - элементы второго множества, целые
 * Ci - элементы третьего множества, целые
 * Di - элементы четвертого множества, целые
 * <num_of_threads> - количество потоков, целые
 */

#include "omp.h"
#include <algorithm>
#include <exception>
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>
#include <ctime>
using namespace std;
vector<vector<int>> vecs;//вектор, состоящий из данных векторов A, B, C, D
int num_of_threads;//количество потоков

/// Метод для параллельного вычисления прямого произведения множеств
/// из вектора vecs
/// \param result - результирующий вектор, состоящий из векторов длины 4 (элементы прямого произведения)
/// \param num_of_threads - заданное количество потоков
void mult(vector<vector<int>> &result, int num_of_threads) {
    int i, j, k, l;
    //директива для разделения итераций циклов между потоками
    double time = omp_get_wtime();
    #pragma omp parallel for num_threads(num_of_threads) collapse(4) private(i,j,k,l)
    for (i = 0; i < vecs[0].size(); ++i) {
        for (j = 0; j < vecs[1].size(); ++j) {
            for (k = 0; k < vecs[2].size(); ++k) {
```

```
            for (l = 0; l < vecs[3].size(); ++l) {
                //критическая секция для записи
                #pragma omp critical
                {
                    result.push_back({vecs[0][i], vecs[1][j], vecs[2][k], vecs[3][l]});
                }
            }
        }
    }
    cout << "Time = " << omp_get_wtime() - time;
}


/// Метод переводящий строку в число
/// \param line - строка
/// \param num - число, в которое нужно записать строку
/// \return - 1, если конвертация прошла успешно, иначе - 0
bool string_to_int(string line, int &num) {
    for (int i = 0; i < line.size(); ++i) {
        if (line[i] > '9' ||
            line[i] < '0')              //если в строке что-то кроме цифр, то это не число
            if (line[i] == '-' && i == 0)//если число отрицательное
                continue;
            else
                return 0;
    }
    num = atoi(line.c_str());
    return 1;
}

/// Метод для считывания входных данных из файла
/// \param path - путь к файлу
/// \param vecs - вектор, в который нужно записать полученные множества
void read_data(string path, vector<vector<int>> &vecs) {
    try {
        ifstream in(path);
        string line{};
        string number{};
        int ind = -1;//индекс множества ([0;3])
        while (getline(in, line)) {
            stringstream strStream(line);
            ind++;
            while (getline(strStream, number, ' ')) {//разделение через пробел
                int num;
                if (!string_to_int(number, num)) {//если в файле есть другие символы
                    //кроме цифр, то он неверный
                    throw exception();
                }
                vecs[ind].push_back(num);
```

```
            }
            if (ind == 3)//если считано 4 множества
                break;
        }
        for (int i = 0; i < vecs.size(); ++i) {
            if (vecs[i].size() == 0)
                throw exception();
        }

        string k;
        in >> k;//считывание количества потоков
        num_of_threads = atoi(k.c_str());
        in.close();
    } catch (const std::exception &err) {
        num_of_threads = 0;
        vecs = vector<vector<int>>(0);
        cout << "Error while reading the file." << endl;
        cout << "Check the correctness of input data" << endl;
        cout << "input data in file: " << endl
             << "A1 A2 ... An" << endl
             << "B1 B2 ... Bm" << endl
             << "C1 C2 ... Ck" << endl
             << "D1 D2 ... Dl" << endl
             << "<num_of_threads>" << endl
             << "where" << endl
             << "Ai - elements of the first multiplicity (integer)" << endl
             << "Bi - elements of the second multiplicity (integer)" << endl
             << "Ci - elements of the third multiplicity (integer)" << endl
             << "Di - elements of the fourth multiplicity (integer)" << endl
             << "<num_of_threads> - number of threads (> 0)" << endl;
    }
}
/// Метод для записи в файл результата выполнения программы
/// \param path - путь к файлу
/// \param result - прямое произведение множеств
void write_res(string path, vector<vector<int>> result) {
    try {
        ofstream out(path);
        if (out.is_open()) {
            out << "{";
            for (int i = 0; i < result.size(); ++i) {
                if (i != 0)
                    out << ",";
                out << "{" << result[i][0] << ", " << result[i][1] << ", "
                    << result[i][2] << ", " << result[i][3] << "}";
                if (i != result.size() - 1)
                    out << endl;
            }
            out << "}";
            out.close();
```

```
        }
    } catch (const exception &err) {
        cout << "Error while writing result into the file.";
    }
}

int main() {
    vector<vector<int>> result; //вектор для результата выполнения программы
    vecs = vector<vector<int>>(4); //вектор для хранения данных множеств
    string pathin; //путь для входных данных
    string pathout; //путь для выходных данных
    cout << "Type input file's path" << endl;
    cin >> pathin; //считывание путей
    cout << "Type output file's path" << endl;
    cin >> pathout; //считывание путей
    read_data(pathin, vecs);
    //рассчет количества элементов в прямом произведении множеств
    int result_num = vecs[0].size() * vecs[1].size() * vecs[2].size() * vecs[3].size();
    if (num_of_threads <= 0) {//количество потоков не может быть <= 0
        cout << "Number of threads must be > 0" << endl;
        cout << "The program will use 2 threads" << endl;
    }
    //потоков не может быть больше чем элементов в декартовом произведении
    if (num_of_threads > result_num) {
        cout << "Too many threads" << endl
             << "The program will use only " << result_num << " threads" << endl;
        num_of_threads = result_num;
    }
    clock_t start_time = clock();//время начала работы потоков
    mult(result, num_of_threads);
    clock_t end_time = clock();//время окончания работы потоков
    //cout << "Time = " << (end_time - start_time) / (double) CLOCKS_PER_SEC;
    write_res(pathout, result);//вывод результата в файл

    return 0;
}
```