Домашнее задание **№4** по Архитектуре вычислительных систем

Выполнил студент 2 курса
группы **БПИ197**
Образовательной программы "Программная инженерия"
Факультета компьютерных наук

Гончаров Егор

Вариант 7

Задача:

7. Вычислить прямое произведение множеств A1, A2, A3… An.
Входные данные: целое положительное число n, множества чисел A1, A2, A3… An, мощности множеств равны между собой и мощность каждого множества больше или равна 1. Количество потоков является входным параметром.

Решение:

Мною был разработан алгоритм, в основе которого лежит принцип "Разделяй и властвуй".
Рассмотрим каждое множество как массив элементов, а лучше не как массив, а как один элемент. Тогда все множества -- это массив элементов, на котором определена операция умножения(прямое произведение). Эта операция ассоциативна. Следовательно неважно в каком порядке мы будем перемножать эти множества, важно только, чтобы они не меняли порядок слева направо. ((a*b)*c = a*(b*c) != b*a*c) В таком случае можно применить подход, так называемой, сортировки слиянием (merge sort) и раздадим эти подзадачи (подотрезки элементов) разным потоком в силу ассоциативности.
Воспользуемся библиотекой openMP для c++. Решение выполнялось на macbook с операционной системой MacOS Catalina версия 10.15.4, для того, чтобы установить библиотеку требуется прописать в командной строке команду homebrew install libomp. Далее нужно настроить XCode и подключить библиотеку omp.h.
Для того, чтобы программа выполнялась параллельно было использовано #pragma omp parallel, для того, чтобы кусок кода не был затронут другим потоком #pragma omp single и для обозначения группы команд, которых необходимо разбить на разные потоки #pragma omp taskgroup.

Код:

```
//
//  main.cpp
//  task4
//
//  Created by Егор Гончаров on 17.11.2020.
//  Copyright © 2020 Егор Гончаров. All rights reserved.
//

#include <iostream>
#include <sstream>
#include <set>
#include <string>
#include <vector>
#include <cstdlib>
#include <thread>
#include <mutex>
#include <omp.h>

#define pb push_back
//#define FILE
#define FAST

using namespace std;

vector<string> ans;
vector< vector<string> > v;
int t;

bool is_digit(string s) {
   for(int i = 0; i < s.size(); ++i)
      if(s[i] < '0' || s[i] > '9')
         return false;
   return true;
}

vector<string> operator * (vector<string> a, vector<string> b) {
   vector<string> ans;
   for(int i = 0; i < a.size(); ++i) {
      for(int j = 0; j < b.size(); ++j) {
         ans.pb(a[i] + ", " + b[j]);
```

```
        }
    }
    return ans;
}
bool f = true;

vector<string> split(string s) {
    string tmp = "";
    vector<string> ans;
    set<string> my_set;
    for(int i = 0; i < s.size(); ++i) {
        if(s[i] == ' ') {
            ans.pb(tmp);
            my_set.insert(tmp);
            f &= is_digit(tmp);
            tmp = "";
            continue;
        }
        tmp += s[i];
    }
    ans.pb(tmp);
    my_set.insert(tmp);
    f &= ans.size() == my_set.size();
    return ans;
}

vector<string> merge(vector<string> a, vector<string> b) {
    return a * b;
} // end of merge()

int getThreadNum() {
    return rand() % t;
}

vector<string> mergeSort(int l, int r)
{
    //cout << l << " " << r << endl;
    if(l == r) {
     return v[l];
    }
    int m=(l+r)/2;
    vector<string> ans1, ans2;
```

```
#pragma omp taskgroup
   {
   ans1 = mergeSort(l, m);
   cout << "Thread["<< omp_get_team_num() + getThreadNum()<< "] " <<
"makes: ";
   for(int i = 0; i < ans1.size(); ++i) {
      cout << "(" << ans1[i] << ")" << ((i == ans1.size() - 1 )? " " : ",");
   }
   cout << endl;


   ans2 = mergeSort(m + 1, r);
   cout << "Thread["<< omp_get_team_num() + getThreadNum() << "] " <<
"makes: ";
   for(int i = 0; i < ans2.size(); ++i) {
      cout << "(" << ans2[i] << ")" << ((i == ans2.size() - 1 )? " " : ",");
   }
   cout << endl;
   }

  #pragma omp taskwait
  vector<string> ansss = merge(ans1, ans2);
  cout << "Thread["<< omp_get_team_num() + getThreadNum() << "] " <<
"makes: ";
  for(int i = 0; i < ansss.size(); ++i) {
     cout << "(" << ansss[i] << ")" << ((i == ansss.size() - 1 )? " " : ",");
  }
  cout << endl;
  return ansss;
}

vector<string> solve(int i, int j) {
   //mt.lock();
   j = j < (int)v.size() ? j : (int)v.size();
   vector<string> local_ans = v[i];
   for(int k = i + 1; k < j; ++k) {
      local_ans = local_ans * v[k];
   }
  // mt.unlock();
   return local_ans;
}
```

```
void print_ans(vector<string> ans) {
   cout << "{ ";
   for(int i = 0; i < ans.size(); ++i) {
      cout << "(" << ans[i] << ")" << ((i == ans.size() - 1 )? " " : ",");
   }
   cout << "} " << endl;
}


int32_t main() {
#ifdef FAST
   ios_base::sync_with_stdio(false);
   cin.tie(nullptr);
   cout.tie(nullptr);
#endif
#ifdef FILE
   freopen("input.txt", "r", stdin);
   freopen("output.txt", "w", stdout);
#endif
   cout << "Enter N -- count of sets: ";
   int n; cin >> n;
   if(n < 0) {
      cout << "Invalid count of sets" << endl;
   }
   string s1;
   std::getline(std::cin, s1);
   v.resize(0);
   cout << "Enter sets like example: Enter 1 set: 1 2 3 4" << endl;
   for(int i = 0; i < n; ++i) {
      cout << "Enter " << i + 1 << " set: ";
      std::getline(std::cin, s1);
      vector<string> s = split(s1);
      if(!f) {
         cout << "Invalid set, you have repetitive digits or you have smt another
digit";
         return 1;
      }
      v.pb(s);
   }
```

```
    cout << "Enter count of threads: ";
    int cnt_threads; cin >> cnt_threads;
    cnt_threads = cnt_threads > n ? n : cnt_threads;
    if(cnt_threads <= 0) {
        cout << "Invalid count of threads" << endl;
        return 1;
    }
    t = cnt_threads;

    srand(static_cast<unsigned int>(time(0)));
    omp_set_num_threads(cnt_threads);
    vector<string> ans2;
   #pragma omp parallel
   #pragma omp single
       ans2 = mergeSort(0, (int)v.size() - 1);

    cout << "Answer: ";
    print_ans(ans2);
    return 0;
}
```