ДСОГЛАСОВАНО

Научный руководитель,  
доцент департамента программной инженерии факультета компьютерных, канд. техн. наук

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Легалов

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

УТВЕРЖДАЮ

**Вариант 12**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 81 01-1 ЛУ**

Исполнитель

cтудент группы БПИ197  
\_\_\_\_\_\_\_\_\_\_\_\_\_/ П. О. Кулешова /

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. В. Шилов

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

**Вариант 12**

**Пояснительная записка**

**RU.17701729.04.01-01 81 01-1**

**Листов 13**

сОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc57747952)

[1.1. Наименование программы 3](#_Toc57747953)

[1.2. Документ, на основе которого ведется разработка 3](#_Toc57747954)

[1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ 4](#_Toc57747955)

[1.1 Назначение разработки 4](#_Toc57747956)

[1.2 Краткая характеристика области применения 4](#_Toc57747957)

[2. Описание программы 5](#_Toc57747958)

[Условие задачи 5](#_Toc57747959)

[Метод решения 5](#_Toc57747960)

[Алгоритм решения 5](#_Toc57747961)

[Замечание 6](#_Toc57747962)

[ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ 8](#_Toc57747963)

[ПРИЛОЖЕНИЕ 1 Код программы 9](#_Toc57747964)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 14](#_Toc57747965)

# ВВЕДЕНИЕ

## Наименование программы

Наименование программы: Разработка многопоточных приложений с использованием OpenMP

Краткое наименование программы: ДЗ

## Документ, на основе которого ведется разработка

http://www.softcraft.ru/edu/comparch/tasks/t04/

# НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

## Назначение разработки

Разработать многопоточную программу для нахождения наибольшей возрастающей последовательности.

## Краткая характеристика области применения

Научно-исследовательская область.

# Описание программы

## Условие задачи

Определить индексы i, j, для которых существует наиболее длинная последовательность А[i] < A[i+1] < A[i+2] < A[i+3] < … < A[j]. Входные данные: массив чисел А, произвольной длины большей 1000. Количество потоков является входным параметром

## Метод решения

Задача решалась с использованием **итеративного параллелизма.** Так как наша программа разбивается на потоки, выполняющие одинаковые подзадачи, работающие в цикле над одной задачей. Все потоки равны. Мы только читаем данные, ничего не записываем.

## Алгоритм решения

1. С помощью #pragma omp parallel for schedule(static) num\_threads(threadNumber),создаём заданное число потоков, разбиваем цикл, который проходит по всем элементам, на практически равные части (schedule(static)).
   1. В цикле, для каждого потока находим наиболее длинную последовательность на его chank
   2. Дожидаемся завершения всех потоков
2. Когда все потоки завершили работу, то ищем наибольшую последовательность:
   1. Проверяем, если после окончания работы, наш поток содержит не обновлённое значение максимальной длины последовательности, то обновляем
   2. Идём по всем потокам и проверяем, хранит ли он более длинную последовательность, чем мы имеем, или начало последовательности следующего потока, является концом нашей. То есть часть возрастающей последовательности попала в наш поток, а часть находится в следующем.
      1. Возможно, что несмотря на тот факт, что конец и начало совпали, следующий поток содержит какую-то новую последовательность. Поэтому обязательно проверяем, что элементы действительно возрастают.
   3. Если часть последовательности оказалась в следующем потоке, то прибавляем «хвост», пока не выполняется это условие
   4. Проверяем, получилась ли итоговая последовательность больше, чем сохранённая. Если да, то перезаписываем результат.
3. Записываем результат поиска в файл.

## Замечание

Считалось, что всегда существует последовательность, состоящая из одного элемента, то есть длины 1. Искалась первая наибольшая последовательность.**3. Формат входных данных**

**Входные данные командной строки**

Программа на вход получает данные из входной строки в следующем формате: *<Путь до файла с тестом>\_<Путь до файла с ответом>\_<Число потоков>*

Значок **\_** означает пробел, то есть входные данные разделены одинарным пробелом.

**Входные данные для тестов**

Тест должен состоять из строки с числом элементов в массиве. На второй строке – элементы массива через пробел.

1. **формат выходных данных**

Текстовый файл, где на первой строке записан индекс начала последовательности, на второй – индекс окончания, на третьей – длина последовательности.

# ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ

1. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению// Единая система программной документации. – М.: ИПК Стандартинформ, 2010.
2. Сайт «SoftCraft». URL: http://www.softcraft.ru/edu/comparch/lect/07-parthread/ [http://www.softcraft.ru] Просмотрено: 01.12.2020
3. Файл «Параллельное программирование на OpenMP». URL: <http://ccfit.nsu.ru/arom/data/openmp.pdf> Просмотрено: 01.12.2020
4. Сайт «Programming Parallel Computers». URL: <http://ppc.cs.aalto.fi/ch3/> Просмотрено: 01.12.2020
5. Файл «Loop Scheduling in OpenMP» (Author: Vivek Kale. University of Southern California/Information Sciences Institute). URL: <https://www.openmp.org/wp-content/uploads/SC17-Kale-LoopSchedforOMP_BoothTalk.pdf> Просмотрено: 01.12.2020
6. Сайт « Блог программиста (программирование и алгоритмы)». URL: <https://pro-prof.com/archives/4335#page_1> Просмотрено: 01.12.2020
7. Сайт «Microsoft». URL: <https://docs.microsoft.com/ru-ru/cpp/build/reference/openmp-enable-openmp-2-0-support?view=msvc-160> Просмотрено: 01.12.2020
8. Сайт «Национальный Открытый Университет». URL: <https://intuit.ru/studies/curriculums/954/courses/232/lecture/6025> Просмотрено: 01.12.2020

# ПРИЛОЖЕНИЕ 1 **Код программы**

#include <iostream>

#include <omp.h>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

/// <summary>

/// Структура данных для потоков

/// </summary>

struct Package {

int i = 0; //Начало последовательности

int j = 0; //Конец последовательности

int t = 0; //Число подряд идущих элементов последовательности

int last = 0; //Индекс последнего элемента, рассматриваемого потоком

};

/// <summary>

/// Функция, которая работает с потоками. Ищет наибольшую последовательность

/// </summary>

/// <param name="threadsArray">Массив данных по потокам</param>

/// <param name="i">Номер элемента массива</param>

/// <param name="num">Число элементов в массиве</param>

/// <param name="array">Массив элементов</param>

void ThreadsFunction(vector<Package>& threadsArray, int i, int num, int\* array);

/// <summary>

/// После отработки потока мы не проверяем, достигли ли максимальной последовательности, поэтому необходимо это сделать

/// </summary>

/// <param name="threadsArray">Массив потоков</param>

/// <param name="i">Номер потока</param>

void LastElement(vector<Package>& threadsArray, int i);

/// <summary>

/// После отработки потоков, необходимо найти наибольшую длину последовательности, так как она,

/// может быть разбита междуу потоками

/// </summary>

/// <param name="threadsArray">Массив потоков</param>

/// <param name="i">Номер потока</param>

/// <param name="finish">Конец наибольшей последовательности</param>

/// <param name="start">Начало наибольшей последовательности</param>

/// <param name="array">Массив чисел</param>

void MaxLength(vector<Package>& threadsArray, int& i, int& finish, int& start, int\* array);

/// <summary>

/// Функция записывает элементы в файл

/// </summary>

/// <param name="answer">Путть к файлу с ответами</param>

/// <param name="start">Индекс начала максимальной последовательности</param>

/// <param name="finish">Индекс окончания максимальной последовательности</param>

void WriteToFile(const string& answer, int start, int finish);

/// <summary>

/// Основная функция

/// </summary>

/// <param name="args">Число аргументов командной строки</param>

/// <param name="argv">Аргументы командной строки</param>

/// <returns></returns>

int main(int args, char\* argv[])

{

const string test = argv[1]; //путь до теста

const string answer = argv[2]; //путь до ответа

const string threds = argv[3]; //число потоков

//создание потока для чтения

ifstream fin(test);

if (!fin.is\_open()) {

throw runtime\_error("IO Exception");

}

int threadNumber = stoi(threds);

int s;

int num;

fin >> num; //число элементов в массиве

int\* array = new int[num];

int w = 0;

//Чтение элементов массива и запись их в массив

while (!fin.eof() && w < num)

{

fin >> s;

array[w] = s;

w++;

}

fin.close(); //закрытие потока

vector<Package> threadsArray(threadNumber);

//Открытие параллельных потоков, с помощью schedule(static) разделены равномерно между потоками.

#pragma omp parallel for schedule(static) num\_threads(threadNumber)

for (int i = 0; i < num; i++)

ThreadsFunction(threadsArray, i, num, array);

//Дожидаемся окончания выполнения всех потоков, преде чем приступим к их анализу

#pragma omp barrier

{

int start = 0;

int finish = 0;

for (int i = 0; i < threadNumber; i++)

LastElement(threadsArray, i);

for (int i = 0; i < threadNumber; i++)

MaxLength(threadsArray, i, finish, start, array);

WriteToFile(answer, start, finish);

delete[] array;

}

return 0;

}

/// <summary>

/// Функция записывает элементы в файл

/// </summary>

/// <param name="answer">Путть к файлу с ответами</param>

/// <param name="start">Индекс начала максимальной последовательности</param>

/// <param name="finish">Индекс окончания максимальной последовательности</param>

void WriteToFile(const string& answer, int start, int finish)

{

fstream out(answer, ios::out);

out << "i = ";

out << start;

out << "\n";

out << "j = ";

out << finish;

out << "\n";

out << "length: ";

out << to\_string(finish - start + 1);

out.close();

}

/// <summary>

/// После отработки потоков, необходимо найти наибольшую длину последовательности, так как она,

/// может быть разбита междуу потоками

/// </summary>

/// <param name="threadsArray">Массив потоков</param>

/// <param name="i">Номер потока</param>

/// <param name="finish">Конец наибольшей последовательности</param>

/// <param name="start">Начало наибольшей последовательности</param>

/// <param name="array">Массив чисел</param>

void MaxLength(vector<Package>& threadsArray, int& i, int& finish, int& start, int\* array)

{

//Проверяем, длине ли сохранённая последовательность, той, что находится в потоке

//Либо нет ли продолжения нашей последовательности в следующем потоке,

//Но при условии, что следующий поток не содержит новую последовательность,

//То есть, пусть дано |1 2 3|-3 -2 -1| и наши потоки данным образом разделили массив

//Тогда, с одной стороны, окончание нашей последовательности, является началом следующей

//Но это две разные последовательности и их нельзя объединять

bool flagNext = i + 1 < threadsArray.size() &&

((threadsArray[i + 1].i == threadsArray[i].j &&

array[threadsArray[i + 1].i] > array[threadsArray[i].j - 1])

|| (threadsArray[i + 1].i == threadsArray[i].j + 1

&& array[threadsArray[i + 1].i] > array[threadsArray[i].j]));

if ((threadsArray[i].j - threadsArray[i].i > finish - start) || flagNext)

{

//Следующая последовательность, не является началом предыдущей

if (!flagNext)

{

start = threadsArray[i].i;

finish = threadsArray[i].j;

}

//Следующая последовательность является началом предыдущей

else

{

int start\_0 = threadsArray[i].i;

int finish\_0 = threadsArray[i].j;

//Пока следующая последовательность является началом предыдущей,

//Мы удлиняем нашу возрастающую последовательность

while ((i + 1 < threadsArray.size() && threadsArray[i].t != 0 &&

((threadsArray[i + 1].i == threadsArray[i].j &&

array[threadsArray[i + 1].i] > array[threadsArray[i].j - 1])

|| (threadsArray[i + 1].i == threadsArray[i].j + 1

&& array[threadsArray[i + 1].i] > array[threadsArray[i].j]))))

{

finish\_0 = threadsArray[i + 1].j;

i++;

}

//Проверяем, стала ли длина нашей новой последовательности больше, чем сохранённая

if (finish\_0 - start\_0 > finish - start)

{

start = start\_0;

finish = finish\_0;

}

}

}

}

/// <summary>

/// После отработки потока мы не проверяем, достигли ли максимальной последовательности, поэтому необходимо это сделать

/// </summary>

/// <param name="threadsArray">Массив потоков</param>

/// <param name="i">Номер потока</param>

void LastElement(vector<Package>& threadsArray, int i)

{

//Если на последней итерации потока, мы получили наилучший результат, но не сохранили его,

//то сейчас перезаписываем значения

if (threadsArray[i].t > (threadsArray[i].j - threadsArray[i].i))

{

threadsArray[i].j = threadsArray[i].last;

threadsArray[i].i = threadsArray[i].last - threadsArray[i].t;

}

}

/// <summary>

/// Функция, которая работает с потоками. Ищет наибольшую последовательность

/// </summary>

/// <param name="threadsArray">Массив данных по потокам</param>

/// <param name="i">Номер элемента массива</param>

/// <param name="num">Число элементов в массиве</param>

/// <param name="array">Массив элементов</param>

void ThreadsFunction(vector<Package>& threadsArray, int i, int num, int\* array)

{

//Записываем номер элемента, с которого начинает работу наш поток

if (threadsArray[omp\_get\_thread\_num()].i == 0)

{

threadsArray[omp\_get\_thread\_num()].i = i;

threadsArray[omp\_get\_thread\_num()].last = i;

}

//Если следующий жлемент больше предыдущего, то увеличиваем счётчик последовательности

if (i + 1 < num && array[i + 1] > array[i])

threadsArray[omp\_get\_thread\_num()].t++;

//Иначе сохраняем полученный результат

else

{

//Если новое значение лучше, чем сохранённое, то перезаписываем

if (threadsArray[omp\_get\_thread\_num()].t > (threadsArray[omp\_get\_thread\_num()].j - threadsArray[omp\_get\_thread\_num()].i))

{

threadsArray[omp\_get\_thread\_num()].j = i; //Здесь, i+1 уже не удовлетворяет

threadsArray[omp\_get\_thread\_num()].i = i - threadsArray[omp\_get\_thread\_num()].t;

}

//В любом случае обнуляем счётчик, так как наша последовательность закончилась

threadsArray[omp\_get\_thread\_num()].t = 0;

}

//Сохраняем значение последнего просмотренного элемента, удовлетворяющего поиску

threadsArray[omp\_get\_thread\_num()].last = i+1; //Здесь i+1 ещё может удовлетворять, так как

//Если бы это был последний (i - последний), то счётчик обнулился и мы им не воспользуемся

//В любом случае, если бы он нас не устраивал, то счётчик был бы равен нулю

//А если это не так, то этот элемент нас устраивает

}

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |