МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №2

по дисциплине «Системное программирование»

**Параллельные алгоритмы в языке C++**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: | студент гр. ФИБ-3301-51-01 |  | / В. Р. Кочкин / |
| Проверил: | к.ф.-м.н. доцент каф. ПМиИ |  | / В. А. Бызов / |

Киров 2022

Цель работы

Получить навыки работы с параллельными алгоритмами в языке C++.

Задания

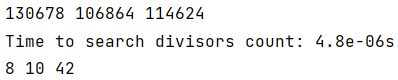
**Задание 1**

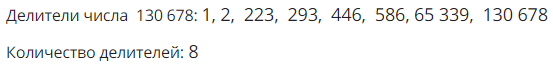
Используя параллельные алгоритмы стандартной библиотеки языка C++, написать программу, вычисляющую для каждого элемента целочисленного вектора количество его делителей. Элементы вектора – случайные натуральные числа из диапазона [105, 106]. Результат должен быть записан в новый вектор.

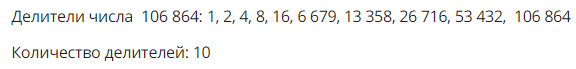
Провести тестирование программы на векторах размера 5 \* 105, 106, 2 \* 106 с различными политиками выполнения. На каждом примере запустить не менее трех раз. В таблицу занести среднее время выполнения на одном примере в секундах. Сравнить результаты. Сделать выводы.

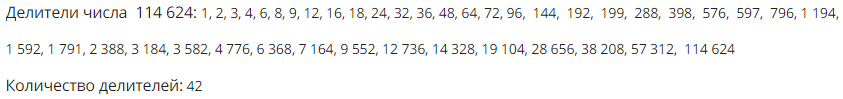
**Полученные результаты**

*Проверка работы алгоритма.* В 1 строке входной вектор, в 3 строке результат нахождения количества делителей.









*Таблица 1– Время вычисления количества делителей*

*элементов целочисленного вектора, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размер вектора | Политика выполнения | | |
| seq | par | par\_unseq |
| 5 \* 105 | 0.76618 | 0.136318 | 0.129423 |
| 106 | 1.53661 | 0.257178 | 0.262655 |
| 2 \* 106 | 3.08626 | 0.515531 | 0.522052 |

Из таблицы видно, что по времени выполнения программы лидируют par и par\_unseq политики выполнения.

Листинг программы приведен в [приложении А задание 1](#_Задание_1.).

**Задание 2**

Используя параллельные алгоритмы стандартной библиотеки языка C++, написать программу, суммирующую элементы вектора.

Указание. Использовать алгоритм std::reduce.

Провести тестирование программы на векторах размера 5 ∙ 108, 109, 2 \* 109 с различными политиками выполнения. На каждом примере запустить не менее трех раз. В таблицу занести среднее время выполнения на одном примере в секундах. Сравнить результаты. Сделать выводы.

**Полученные результаты**

*Проверка работы алгоритма.* В 1 строке входной вектор, в 3 строке результат суммирования.

Изображение выглядит как текст, оранжевый

Автоматически созданное описание

*Таблица 2 – Время выполнения алгоритма суммирования   
элементов вектора, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размер вектора | Политика выполнения | | |
| seq | par | par\_unseq |
| 5 \* 108 | 0.0859548 | 0.051699 | 0.0522486 |
| 109 | 0.166121 | 0.107143 | 0.104839 |
| 2 \* 109 | 0.432894 | 0.208253 | 0.224499 |

Из таблицы видно, что по времени выполнения программы лидируют par и par\_unseq политики выполнения.

Листинг программы приведен в [приложении А задание 2](#_Задание_2.).

**Задание 3**

Используя параллельные алгоритмы стандартной библиотеки языка C++, реализовать классический алгоритм умножения матриц.

Указание. Алгоритмы использовать для вычисления скалярного произведения.

Провести тестирование программы на матрицах размера 512, 1024, 2048 с различными политиками выполнения. На каждом примере запустить не менее трех раз. В таблицу занести среднее время выполнения на одном примере в секундах. Сравнить результаты с результатами лабораторной работы 1. Сделать выводы.

**Полученные результаты**

*Проверка работы алгоритма.* Первая и вторая матрицы – входные. Третья матрица - результат.



Изображение выглядит как часы

Автоматически созданное описание

*Таблица 3 – Время выполнения классического алгоритма   
умножения матриц, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы | Политика выполнения | | |
| seq | par | par\_unseq |
| 512 | 0.0176926 | 1.28419 | 1.37147 |
| 1024 | 0.138595 | 7.73854 | 7.35719 |
| 2048 | 2.02386 | 31.7538 | 31.4294 |

Из таблицы видно, что по времени выполнения программы лидирует seq политика выполнения.

Листинг программы приведен в [приложении А задание 3](#_Задание_3.).

**Задание 4**

Используя параллельные алгоритмы стандартной библиотеки языка C++,

написать программу, сортирующую массив действительных чисел.

Протестировать на массивах разной размерности (3–5 вариантов) со всеми

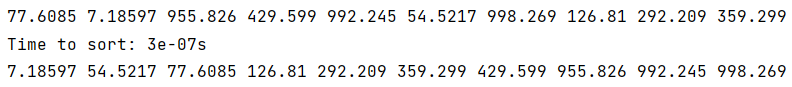
политиками выполнения. Замерить время выполнения. Размеры массивов

подобрать таким образом, чтобы нагрузить систему. Сравнить результаты.

Результаты должны быть представлены в удобном, понятном и наглядном виде.

**Полученные результаты**

*Проверка работы алгоритма.* В 1 строке входной вектор, в 3 строке результат сортировки.



*Таблица 4 – Время выполнения алгоритма сортировки  
массива действительных чисел, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размер массива | Политика выполнения | | |
| seq | par | par\_unseq |
| 106 | 0.0699026 | 0.0141052 | 0.0141912 |
| 107 | 0.655288 | 0.143609 | 0.127869 |
| 108 | 6.24055 | 1.39541 | 1.39876 |
| 5\*108 | 30.451 | 6.73997 | 7.14446 |
| 109 | 61.1282 | 14.401 | 14.4048 |

Из таблицы видно, что по времени выполнения программы лидируют par и par\_unseq политики выполнения.

Листинг программы приведен в [приложении А задание 4](#_Задание_4.).

Вывод

В ходе лабораторной работы я получил навыки работы с параллельными алгоритмами в языке C++. Были произведены замеры времени выполнения программ и на их основании сделаны выводы. Все тесты проводились на процессоре AMD Ryzen 5 3600 6/12 4.2ГГц.

# **Приложения**

## **Приложение А. Листинги программ**

### Задание 1.

#include <iostream>  
#include <vector>  
#include <ctime>  
#include <chrono>  
#include <execution>  
  
void fill\_vector(std::vector<int> &, int, int, int);  
void print\_vector(const std::vector<int>&);  
int calculate\_divisor\_count(int);  
double calculate\_search\_divisor\_count\_time(std::vector<int>, std::vector<int> &);  
  
const int SIZE = 500000;  
const int LEFT\_BORDER = 100000;  
const int RIGHT\_BORDER = 1000000 + 1;  
  
int main()  
{  
 srand(time(nullptr));  
  
 std::vector<int> vector;  
 fill\_vector(vector, SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
  
 */\*print\_vector(vector);  
 std::cout << std::endl;\*/* std::vector<int> result(vector.size());  
 double time = calculate\_search\_divisor\_count\_time(vector, result);  
 std::cout << "Time to search divisors count: " << time << "s" << std::endl;  
  
 */\*print\_vector(result);\*/* return 0;  
}  
  
void fill\_vector(std::vector<int> &vector, int size, int left\_border, int right\_border)  
{  
 for (size\_t i = 0; i < size; i++) {  
 vector.push\_back(rand() % right\_border + left\_border);  
 }  
}  
  
void print\_vector(const std::vector<int>& vector)  
{  
 for (int i : vector) {  
 std::cout << i << " ";  
 }  
}  
  
int calculate\_divisor\_count(int n)  
{  
 int count = 0;  
 for (int i = 1; i \* i <= n; i++) {  
 if (n % i == 0) {  
 if (i \* i != n) {  
 count++;  
 }  
 count++;  
 }  
 }  
  
 return count;  
}  
  
double calculate\_search\_divisor\_count\_time(std::vector<int> vector, std::vector<int> &result)  
{  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 std::transform(std::execution::seq, vector.begin(), vector.end(), result.begin(), calculate\_divisor\_count);  
   
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 return diff.count();  
}

### Задание 2.

#include <iostream>  
#include <vector>  
#include <ctime>  
#include <chrono>  
#include <execution>  
#include <algorithm>  
  
void fill\_vector(std::vector<int>&, int, int, int);  
void print\_vector(const std::vector<int>&);  
double calculate\_sum\_time(std::vector<int>, long long &);  
  
const long SIZE = 500000000;  
const int LEFT\_BORDER = 0;  
const int RIGHT\_BORDER = 1 + 1;  
  
int main()  
{  
 srand(time(nullptr));  
  
 std::vector<int> vector;  
 fill\_vector(vector, SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
  
 */\*print\_vector(vector);  
 std::cout << std::endl;\*/* long long result = 0;  
 double time = calculate\_sum\_time(vector, result);  
 std::cout << "Time to sum vector: " << time << "s" << std::endl;  
  
 */\*std::cout << "Result: " << result << std::endl;\*/* return 0;  
}  
  
void fill\_vector(std::vector<int>& vector, int size, int left\_border, int right\_border)  
{  
 for (size\_t i = 0; i < size; i++) {  
 vector.push\_back(rand() % right\_border + left\_border);  
 }  
}  
  
void print\_vector(const std::vector<int>& vector)  
{  
 for (int i : vector) {  
 std::cout << i << " ";  
 }  
}  
  
double calculate\_sum\_time(std::vector<int> vector, long long &result)  
{  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 result = std::reduce(std::execution::seq, vector.begin(), vector.end());  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 return diff.count();  
}

### Задание 3.

#include <iostream>  
#include <cstdlib>  
#include <chrono>  
#include <vector>  
#include <ctime>  
#include <execution>  
  
std::vector<std::vector<int>> create\_matrix(int);  
void fill\_matrix(std::vector<std::vector<int>>&, int, int);  
void print\_matrix(std::vector<std::vector<int>>&);  
void transposition(std::vector<std::vector<int>>&);  
std::vector<std::vector<int>> multiply\_matrix(std::vector<std::vector<int>>, std::vector<std::vector<int>>);  
double calculate\_multiply\_time(std::vector<std::vector<int>>, std::vector<std::vector<int>>);  
  
const int SIZE = 512;  
const int LEFT\_BORDER = 0;  
const int RIGHT\_BORDER = 10;  
  
int main()  
{  
 srand(time(nullptr));  
  
 std::vector<std::vector<int>> matrix\_1 = create\_matrix(SIZE);  
 std::vector<std::vector<int>> matrix\_2 = create\_matrix(SIZE);  
 fill\_matrix(matrix\_1, LEFT\_BORDER, RIGHT\_BORDER);  
 fill\_matrix(matrix\_2, LEFT\_BORDER, RIGHT\_BORDER);  
  
 */\*print\_matrix(matrix\_1);  
 std::cout << std::endl;  
 print\_matrix(matrix\_2);  
 std::cout << std::endl;\*/* transposition(matrix\_2); *// Транспонирование матрицы B для корректной работы transform\_reduce* double time = calculate\_multiply\_time(matrix\_1, matrix\_2);  
 std::cout << "Time to multiply: " << time << "s" << std::endl;  
}  
  
std::vector<std::vector<int>> create\_matrix(int size)  
{  
 std::vector<std::vector<int>> matrix;  
 matrix.reserve(size);  
 for (int i = 0; i < size; i++) {  
 matrix.emplace\_back();  
 }  
  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 matrix[i].push\_back(0);  
 }  
 }  
  
 return matrix;  
}  
  
void fill\_matrix(std::vector<std::vector<int>> &matrix, int left\_border, int right\_border)  
{  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix.size(); j++) {  
 matrix[i][j] = rand() % right\_border - left\_border;  
 }  
 }  
}  
  
void print\_matrix(std::vector<std::vector<int>> &matrix)  
{  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix.size(); j++) {  
 std::cout << matrix[i][j] << " ";  
 }  
 std::cout << std::endl;  
 }  
}  
  
void transposition(std::vector<std::vector<int>>& matrix)  
{  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = i; j < matrix[i].size(); j++) {  
 int t = matrix[i][j];  
 matrix[i][j] = matrix[j][i];  
 matrix[j][i] = t;  
 }  
 }  
}  
  
std::vector<std::vector<int>> multiply\_matrix(std::vector<std::vector<int>> matrix\_1, std::vector<std::vector<int>> matrix\_2)  
{  
 int size = matrix\_1.size();  
 std::vector<std::vector<int>> result(size, std::vector<int>(size));  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 */\*  
 \* Cкалярное произведение векторов matrix\_1[i] и matrix\_2[j]  
 \* Результат записывается в [i][j] ячейку результирующей матрицы  
 \*/* result[i][j] = std::transform\_reduce(std::execution::seq, matrix\_1[i].begin(), matrix\_1[i].end(), matrix\_2[j].begin(), 0);  
 }  
 }  
  
 return result;  
}  
  
double calculate\_multiply\_time(std::vector<std::vector<int>> matrix\_1, std::vector<std::vector<int>> matrix\_2)  
{  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 std::vector<std::vector<int>> result = multiply\_matrix(matrix\_1, matrix\_2);  
   
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 */\*print\_matrix(result);  
 std::cout << std::endl;\*/* return diff.count();  
}

### Задание 4.

#include <iostream>  
#include <cstdlib>  
#include <chrono>  
#include <ctime>  
#include <vector>  
#include <execution>  
#include <algorithm>  
  
double\* create\_random\_array(int, int, int);  
void print\_array(double\*, int);  
double calculate\_sort\_time(double\*, int);  
  
const int SIZE = 1000000;  
const int LEFT\_BORDER = 0;  
const int RIGHT\_BORDER = 1000 + 1;  
  
int main()  
{  
 srand(time(nullptr));  
  
 double\* array = create\_random\_array(SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
  
 */\*print\_array(array, SIZE);  
 std::cout << std::endl;\*/* double time = calculate\_sort\_time(array, SIZE);  
 std::cout << "Time to sort: " << time << "s" << std::endl;  
  
 */\*print\_array(array, SIZE);  
 std::cout << std::endl;\*/*}  
  
double get\_random\_number(int min, int max)  
{  
 static const double fraction = 1.0 / (static\_cast<double>(**RAND\_MAX**) + 1.0);  
 return static\_cast<double>(rand() \* fraction \* (max - min + 1) + min);  
}  
  
double\* create\_random\_array(int size, int left\_border, int right\_border)  
{  
 auto \*result = new double[size];  
 for (int i = 0; i < size; i++) {  
 result[i] = get\_random\_number(left\_border, right\_border);  
 }  
  
 return result;  
}  
  
void print\_array(double\* array, int size)  
{  
 for (int i = 0; i < size; i++) {  
 std::cout << array[i] << " ";  
 }  
}  
  
double calculate\_sort\_time(double\* array, int size)  
{  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 std::sort(std::execution::seq, array, array + size);  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 return diff.count();  
}