**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**СЕРГИЕНКО ЛЕВ ЭДУАРДОВИЧ**

Отчет по
Лабораторная работа 12
Параллельные вычисления в C++
Завершение потока
Автоматическое распараллеливание кода, использующего стандартные алгоритмы

**Преподаватель**

***Кондратьева О.М.***

**2025**

## Задание 1

Известная задача – улучшить отзывчивость интерфейса
Вариант 1 (до C++20):
заведите флаг завершения потока, защитите доступ к нему;
завершайте поток по флагу;
«внешний» поток после изменения флага вызывает std::join().
Вариант 2 (C++20):
В стандарте C++ 20 [1] появился усовершенствованный класс для поддержки потоков std::jthread.

### Тексты программ

```
#include <atomic>
#include <chrono>
#include <iostream>
#include <thread>

static std::atomic<bool> terminationSignal{false};

void workerRoutine()
{
    while (!terminationSignal.load())
    {
        std::cout << "Working...\n";
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
    }
    std::cout << "Background task stopping\n";
}

int main()
{
    std::thread backgroundWorker{workerRoutine};

    std::this_thread::sleep_for(std::chrono::seconds(3));
    std::cout << "Stop requested\n";
    terminationSignal.store(true);

    backgroundWorker.join();
    std::cout << "Worker thread has ended\n";
    return 0;
}
```

```cpp
#include <chrono>
#include <iostream>
#include <stop_token>
#include <thread>

void workerRoutine(std::stop_token stopToken)
{
    while (!stopToken.stop_requested())
    {
        std::cout << "Working...\n";
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
    }
    std::cout << "Background task stopping\n";
}

int main()
{
    std::jthread backgroundWorker{workerRoutine};

    std::this_thread::sleep_for(std::chrono::seconds(3));
    std::cout << "Stop requested\n";

    backgroundWorker.request_stop();
    backgroundWorker.join();
    std::cout << "Worker thread has ended\n";
    return 0;
}
```

## Копии экранов

```
lev@redmibook-15 /m/c/U/l/D/B/3/p/lab12 (main)> make run1.1
g++ -std=c++17 -O2 -Wall -Wextra -pthread t1.1.cpp -o bin/t1.1
./bin/t1.1
Working...
Working...
Working...
Working...
Working...
Working...
Stop requested
Background task stopping
Worker thread has ended
```

```
lev@redmibook-15 /m/c/U/l/D/B/3/p/lab12 (main)> make run1.2
g++ -std=c++20 -O2 -Wall -Wextra -pthread t1.2.cpp -o bin/t1.2
./bin/t1.2
Working...
Working...
Working...
Working...
Working...
Working...
Working...
Stop requested
Background task stopping
Worker thread has ended
```

## Задание 2

В [2] представлены подробный пример и список алгоритмов. Ставим эксперименты со стандартными алгоритмами.
Выберите три алгоритма, которые можно автоматически распараллелить.
Возьмите контейнер большой размерности и обработайте его с использованием выбранных алгоритмов.
Выполните вычислительные эксперименты и определите эффективность параллельной реализации.

### Текст программы

```cpp
#include <algorithm>
#include <chrono>
#include <execution>
#include <iostream>
#include <random>
#include <vector>

static bool isOdd(int value)
{
    return (value % 2) != 0;
}

int main()
{
    constexpr size_t ELEMENT_COUNT = 100'000'000;
    std::vector<int> values(ELEMENT_COUNT);

    std::mt19937_64 rng{std::random_device{}()};
```

```
std::uniform_int_distribution<int> dist(0, 1'000'000);
std::generate(std::execution::par, values.begin(), values.end(),
              [&]()
              { return dist(rng); });

// measure single-threaded sort
auto t_start = std::chrono::high_resolution_clock::now();
std::sort(std::execution::seq, values.begin(), values.end());
auto t_end = std::chrono::high_resolution_clock::now();
std::cout << "Single-threaded sort took: "
          << std::chrono::duration<double>(t_end - t_start).count()
          << " seconds\n";

// measure single-threaded max_element
t_start = std::chrono::high_resolution_clock::now();
auto max_it_seq = std::max_element(std::execution::seq,
                                   values.begin(), values.end());
t_end = std::chrono::high_resolution_clock::now();
std::cout << "Single-threaded max_element took: "
          << std::chrono::duration<double>(t_end - t_start).count()
          << " seconds\n"
          << "Maximum (seq): " << *max_it_seq << "\n";

// measure single-threaded count_if
t_start = std::chrono::high_resolution_clock::now();
auto odd_count_seq = std::count_if(std::execution::seq,
                                   values.begin(), values.end(),
                                   isOdd);
t_end = std::chrono::high_resolution_clock::now();
std::cout << "Single-threaded count_if(isOdd) took: "
          << std::chrono::duration<double>(t_end - t_start).count()
          << " seconds\n"
          << "Odd count (seq): " << odd_count_seq << "\n\n";

// parallel versions
t_start = std::chrono::high_resolution_clock::now();
std::sort(std::execution::par, values.begin(), values.end());
t_end = std::chrono::high_resolution_clock::now();
std::cout << "Parallel sort took: "
          << std::chrono::duration<double>(t_end - t_start).count()
          << " seconds\n";

t_start = std::chrono::high_resolution_clock::now();
auto max_it_par = std::max_element(std::execution::par,
                                   values.begin(), values.end());
t_end = std::chrono::high_resolution_clock::now();
std::cout << "Parallel max_element took: "
          << std::chrono::duration<double>(t_end - t_start).count()
          << " seconds\n"
          << "Maximum (par): " << *max_it_par << "\n";
```

```
    t_start = std::chrono::high_resolution_clock::now();
    auto odd_count_par = std::count_if(std::execution::par,
                                       values.begin(), values.end(),
                                       isOdd);
    t_end = std::chrono::high_resolution_clock::now();
    std::cout << "Parallel count_if(isOdd) took: "
              << std::chrono::duration<double>(t_end - t_start).count()
              << " seconds\n"
              << "Odd count (par): " << odd_count_par << "\n";

    return 0;
}
```

## Результаты экспериментов

count_if

| Размерность задачи | Время выполнения последовательной, с | Время выполнения | Ускорение |
|---|---|---|---|
| 10 000 000 | 0,0136579 | 0,0136877 | 0,9978228629 |
| 100 000 000 | 0,139728 | 0,164643 | 0,8486725825 |
| 1 000 000 000 | 1,43099 | 1,44568 | 0,9898386918 |

sort

| Размерность задачи | Время выполнения последовательной, с | Время выполнения | Ускорение |
|---|---|---|---|
| 10 000 000 | 0,603949 | 0,13421 | 4,500029804 |
| 100 000 000 | 5,98952 | 0,939892 | 6,372561954 |
| 1 000 000 000 | 57,8751 | 10,9693 | 5,276097837 |

max

| Размерность задачи | Время выполнения последовательной, с | Время выполнения | Ускорение |
|---|---|---|---|
| 10 000 000 | 0,0162682 | 0,0120484 | 1,350237376 |
| 100 000 000 | 0,161653 | 0,0747085 | 2,163783238 |
| 1 000 000 000 | 1,72663 | 0,561405 | 3,075551518 |