МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

СЕРГИЕНКО ЛЕВ ЭДУАРДОВИЧ

Отчет по
Лабораторная работа 13
Параллельные вычисления в C++
Распараллеливание кода

Преподаватель

Кондратьева О.М.

2025

Задание 1

Поиск значения максимального элемента вектора. Реализуйте параллельные программы для решения задачи в вариантах:

- в стиле простой реализация параллельной версии std::accumulate [4, Раздел 2.4.];
- по методу «Разделяй и властвуй»;
- используя стандартные алгоритмы с различными политиками выполнения;
- технология OpenMP, различные варианты.

Тексты программ

в стиле простой реализация параллельной версии std::accumulate

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <future>
#include <numeric>
#include <random>
#include <chrono>

using Clock = std::chrono::high_resolution_clock;
using Duration = std::chrono::duration<double>;

template <typename It>
int findMaximum(It first, It last)
{
    auto count = std::distance(first, last);
    if (count < 2000)
    {
        return *std::max_element(first, last);
    }

    It mid = first + count / 2;

    auto rightFuture = std::async(std::launch::async, findMaximum<It>, mid,
last);

    int leftMax = findMaximum(first, mid);
    int rightMax = rightFuture.get();

    return std::max(leftMax, rightMax);
}

int main()
```

```
{
    std::vector<size_t> testSizes = {1'000'000, 10'000'000, 100'000'000};

    std::mt19937_64 rng(12345);

    for (auto n : testSizes)
    {
        std::vector<int> data(n);
        std::iota(data.begin(), data.end(), 1);
        std::shuffle(data.begin(), data.end(), rng);

        auto t0 = Clock::now();
        int maximum = findMaximum(data.begin(), data.end());
        auto t1 = Clock::now();

        Duration elapsed = t1 - t0;
        std::cout << "N = " << n
                  << "  | Max = " << maximum
                  << "  | Time = " << elapsed.count() << " s\n";
    }

    return 0;
}
```

по методу «Разделяй и властвуй»

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <future>
#include <numeric>
#include <random>
#include <chrono>

using Clock = std::chrono::high_resolution_clock;
using Seconds = std::chrono::duration<double>;

int computeMaxDC(const std::vector<int> &arr, size_t left, size_t right, size_t
cutOff = 2000)
{
    size_t span = right - left;
    if (span <= cutOff)
    {
        return *std::max_element(arr.begin() + left, arr.begin() + right);
    }

    size_t mid = left + span / 2;
    auto rightFut = std::async(std::launch::async, computeMaxDC,
                               std::cref(arr), mid, right, cutOff);
    int leftMax = computeMaxDC(arr, left, mid, cutOff);
```

```
    int rightMax = rightFut.get();
    return std::max(leftMax, rightMax);
}

int main()
{
    std::vector<size_t> sizes = {1'000'000, 10'000'000, 100'000'000};
    std::mt19937_64 rng(12345);

    for (auto N : sizes)
    {
        std::vector<int> data(N);
        std::iota(data.begin(), data.end(), 1);
        std::shuffle(data.begin(), data.end(), rng);

        auto tStart = Clock::now();
        int maximum = computeMaxDC(data, 0, data.size());
        auto tEnd = Clock::now();

        Seconds elapsed = tEnd - tStart;
        std::cout << "Size = " << N
                  << "  | Max = " << maximum
                  << "  | Time = " << elapsed.count() << " s\n";
    }

    return 0;
}
```

используя стандартные алгоритмы с различными политиками выполнения

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

using Clock = std::chrono::high_resolution_clock;
using Seconds = std::chrono::duration<double>;

enum class Policy
{
    Seq,
    Par,
    ParUnseq
};

const char *policyName(Policy p)
```

```
{
    switch (p)
    {
    case Policy::Seq:
        return "seq";
    case Policy::Par:
        return "par";
    case Policy::ParUnseq:
        return "par_unseq";
    }
    return "";
}

template <typename It>
int maxWithPolicy(Policy pol, It first, It last)
{
    switch (pol)
    {
    case Policy::Seq:
        return *std::max_element(std::execution::seq, first, last);
    case Policy::Par:
        return *std::max_element(std::execution::par, first, last);
    case Policy::ParUnseq:
        return *std::max_element(std::execution::par_unseq, first, last);
    }
    return *std::max_element(first, last);
}

int main()
{
    std::vector<size_t> sizes = {1'000'000, 10'000'000, 100'000'000};
    std::mt19937_64 rng(12345);

    for (auto N : sizes)
    {
        std::vector<int> data(N);
        std::iota(data.begin(), data.end(), 1);
        std::shuffle(data.begin(), data.end(), rng);

        std::cout << "---- N = " << N << " ----\n";
        for (Policy pol : {Policy::Seq, Policy::Par, Policy::ParUnseq})
        {
            auto t0 = Clock::now();
            int maximum = maxWithPolicy(pol, data.begin(), data.end());
            auto t1 = Clock::now();

            Seconds elapsed = t1 - t0;
            std::cout << policyName(pol)
                      << "  | Max = " << maximum
                      << "  | Time = " << elapsed.count() << " s\n";
        }
```

```
        std::cout << "\n";
    }

    return 0;
}
```

технология OpenMP

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <random>
#include <chrono>
#include <omp.h>

using Clock = std::chrono::high_resolution_clock;
using Seconds = std::chrono::duration<double>;

int main()
{
    std::vector<size_t> sizes = {1'000'000, 10'000'000, 100'000'000};
    std::mt19937_64 rng(12345);

    for (auto N : sizes)
    {
        std::vector<int> data(N);
        std::iota(data.begin(), data.end(), 1);
        std::shuffle(data.begin(), data.end(), rng);

        int globalMax = std::numeric_limits<int>::min();
        auto t0 = Clock::now();
#pragma omp parallel for reduction(max : globalMax)
        for (size_t i = 0; i < data.size(); ++i)
        {
            globalMax = std::max(globalMax, data[i]);
        }
        auto t1 = Clock::now();

        Seconds elapsed = t1 - t0;
        std::cout << "Size = " << N
                  << "  | Max = " << globalMax
                  << "  | Time = " << elapsed.count() << " s\n";
    }

    return 0;
}
```

Результаты экспериментов

Размерность задачи	вариант 1, с	вариант 2, с	вариант 3, с	вариант 4, с
1000000	0,0430294	0,0519336	0,000453375	0,000844099
10000000	0,308429	0,323379	0,00501626	0,00148912
100000000	2,9616	2,91964	0,0551582	0,0115734

```
lev@redmibook-15 /m/c/U/l/D/B/3/p/lab13 (main)> make run1
g++ -std=c++17 -O2 -pthread t1.cpp -o bin/t1
./bin/t1
N = 1000000  | Max = 1000000  | Time = 0.0430294 s
N = 10000000  | Max = 10000000  | Time = 0.308429 s
N = 100000000  | Max = 100000000  | Time = 2.9616 s
lev@redmibook-15 /m/c/U/l/D/B/3/p/lab13 (main)> make run2
g++ -std=c++17 -O2 -pthread t2.cpp -o bin/t2
./bin/t2
Size = 1000000  | Max = 1000000  | Time = 0.0519336 s
Size = 10000000  | Max = 10000000  | Time = 0.323379 s
Size = 100000000  | Max = 100000000  | Time = 2.91964 s
lev@redmibook-15 /m/c/U/l/D/B/3/p/lab13 (main)> make run3
g++ -std=c++17 -O2 -pthread t3.cpp -o bin/t3
./bin/t3
---- N = 1000000 ----
seq  | Max = 1000000  | Time = 0.000453375 s
par  | Max = 1000000  | Time = 0.000450672 s
par_unseq  | Max = 1000000  | Time = 0.000463524 s

---- N = 10000000 ----
seq  | Max = 10000000  | Time = 0.00509053 s
par  | Max = 10000000  | Time = 0.00501379 s
par_unseq  | Max = 10000000  | Time = 0.00501626 s

---- N = 100000000 ----
seq  | Max = 100000000  | Time = 0.0596405 s
par  | Max = 100000000  | Time = 0.0551582 s
par_unseq  | Max = 100000000  | Time = 0.0580516 s

lev@redmibook-15 /m/c/U/l/D/B/3/p/lab13 (main)> make run4
g++ -std=c++17 -O2 -pthread -fopenmp t4.cpp -o bin/t4
./bin/t4
Size = 1000000  | Max = 1000000  | Time = 0.000844099 s
Size = 10000000  | Max = 10000000  | Time = 0.00148912 s
Size = 100000000  | Max = 100000000  | Time = 0.0115734 s
lev@redmibook-15 /m/c/U/l/D/B/3/p/lab13 (main)>
```