МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

СЕРГИЕНКО ЛЕВ ЭДУАРДОВИЧ

Отчет по
Лабораторная работа 11
Параллельные вычисления в C++ Независимые подзадачи

Преподаватель

Кондратьева О.М.

2025

Задание 1

Суммировать элементы последовательного контейнера по Энтони Уильямсу [2, Листинг 2.9. Простейшая параллельная версия std::accumulate, стр. 60].

Текст программы

```
#include <iostream>
#include <vector>
#include <thread>
#include <numeric>
#include <chrono>
#include <functional>

template <typename Iter, typename Val>
class BlockSummer
{
public:
    void operator()(Iter begin, Iter finish, Val &output)
    {
        output = std::accumulate(begin, finish, output);
    }
};

template <typename Iter, typename Val>
Val multiThreadSum(Iter begin, Iter finish, Val initial, size_t
workerCount)
{
    size_t dataLength = std::distance(begin, finish);

    if (dataLength == 0)
    {
        return initial;
    }

    // Calculate chunk size for each thread
    size_t chunkSize = dataLength / workerCount;

    // Store partial results and thread objects
    std::vector<Val> partialSums(workerCount);
    std::vector<std::thread> workers(workerCount - 1);

    // Distribute work among threads
    Iter chunkBegin = begin;
    for (size_t i = 0; i < (workerCount - 1); ++i)
```

```
    {
        Iter chunkEnd = chunkBegin;
        std::advance(chunkEnd, chunkSize);

        workers[i] = std::thread(
            BlockSummer<Iter, Val>(),
            chunkBegin,
            chunkEnd,
            std::ref(partialSums[i]));

        chunkBegin = chunkEnd;
    }

    BlockSummer<Iter, Val>()(chunkBegin, finish, partialSums[workerCount
- 1]);

    for (auto &worker : workers)
    {
        worker.join();
    }

    // Combine partial results
    return std::accumulate(partialSums.begin(), partialSums.end(),
initial);
}

int main()
{
    const std::vector<unsigned long> arraySizes = {10000000, 100000000,
1000000000};

    const std::vector<size_t> threadCounts = {1, 2, 4, 8};

    for (const auto &arraySize : arraySizes)
    {
        std::cout << "Array size: " << arraySize << std::endl;

        std::vector<int> dataset(arraySize);
        std::iota(dataset.begin(), dataset.end(), 1);

        for (const auto &threadCount : threadCounts)
        {
            auto startTime = std::chrono::high_resolution_clock::now();

            long long finalSum = multiThreadSum(
                dataset.begin(),
```

```
                dataset.end(),
                0LL,
                threadCount);

            auto endTime = std::chrono::high_resolution_clock::now();
            auto elapsedTime = std::chrono::duration<double>(endTime -
startTime);

            std::cout << "  Workers: " << threadCount
                      << " | Duration: " << elapsedTime.count() << "s"
                      << " | Sum: " << finalSum << std::endl;
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Таблица с результатами экспериментов

Размерность задачи	Время выполнения последовательной, с	2 потока			4 потока			8 потока		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
10 000 000	0,004526	0,003526	1,283566	0,641783	0,002808	1,611620	0,402905	0,003558	1,272106	0,159013
100 000 000	0,059299	0,028214	2,101705	1,050852	0,021055	2,816314	0,704078	0,016610	3,570025	0,446253
1 000 000 000	0,964634	0,301879	3,195432	1,597716	0,137448	7,018174	1,754543	0,095266	10,12561	1,26570

Задание 2

Параллельный алгоритм для вычисления определенного интеграла.
• Построить параллельный вычислитель определенного интеграла методом левых прямоугольников. Входные данные (функция и отрезок интегрирования) заданы корректно.
• Провести вычислительные экспериментов для различных функций (3 функции), размерностей задачи (4 размерности), количества потоков (2 потока и 4 потока).
Требование: В стиле Параллельного суммирования по Энтони Уильямсу.

Текст программы

```
#include <iostream>
#include <vector>
#include <thread>
#include <functional>
#include <chrono>
#include <cmath>

double eq_squared(double val) { return val * val; }
double eq_sine(double val) { return std::sin(val); }
double eq_gaussian(double val) { return std::exp(-val * val); }

struct calc_segment
{
    void compute(std::function<double(double)> expression,
                 double start_point, double step_size, size_t first_idx, size_t
last_idx,
                 double &segment_value)
    {
        double accumulator = 0.0;
        for (size_t idx = first_idx; idx < last_idx; ++idx)
        {
            double position = start_point + idx * step_size;
            accumulator += expression(position);
        }
        segment_value = accumulator * step_size;
    }
};

double calculate_numerical_approximation(std::function<double(double)>
expression,
                                         double lower_limit, double upper_limit,
                                         size_t sample_count,
                                         size_t worker_count)
{
    double step_size = (upper_limit - lower_limit) / sample_count;

    std::vector<double> partial_results(worker_count);
    std::vector<std::thread> worker_threads(worker_count - 1);

    size_t samples_per_worker = sample_count / worker_count;

    // Create worker threads
    for (size_t worker_id = 0; worker_id < worker_count - 1; ++worker_id)
    {
        size_t start_sample = worker_id * samples_per_worker;
        size_t end_sample = start_sample + samples_per_worker;

        worker_threads[worker_id] = std::thread(
            [&](size_t start, size_t end, size_t id)
```

```
            {
                calc_segment().compute(expression, lower_limit, step_size,
                                       start, end, partial_results[id]);
            },
            start_sample, end_sample, worker_id);
    }

    // Process final segment in current thread
    size_t final_start = (worker_count - 1) * samples_per_worker;
    size_t final_end = sample_count;
    calc_segment().compute(expression, lower_limit, step_size,
                           final_start, final_end, partial_results[worker_count
- 1]);

    for (auto &thread : worker_threads)
        thread.join();

    double final_result = 0.0;
    for (double partial : partial_results)
        final_result += partial;

    return final_result;
}

int main()
{
    std::vector<std::string> formula_labels = {"x^2", "sin(x)", "exp(-x^2)"};
    std::vector<std::function<double(double)>> formulas = {eq_squared, eq_sine,
eq_gaussian};
    std::vector<size_t> sample_counts = {10000000, 100000000, 1000000000};
    std::vector<size_t> worker_counts = {1, 2, 4};

    double lower_bound = 0.0, upper_bound = 1.0;

    for (size_t formula_idx = 0; formula_idx < formulas.size(); ++formula_idx)
    {
        std::cout << "\nFormula: " << formula_labels[formula_idx] << "\n";
        for (size_t samples : sample_counts)
        {
            for (size_t workers : worker_counts)
            {
                auto time_begin = std::chrono::high_resolution_clock::now();

                double answer =
calculate_numerical_approximation(formulas[formula_idx],
                                                                  lower_bound,
upper_bound,
                                                                  samples,
workers);

                auto time_end = std::chrono::high_resolution_clock::now();
```

```
                std::chrono::duration<double> elapsed = time_end - time_begin;

                std::cout << "  Samples=" << samples << ", Workers=" << workers
                          << ", Value=" << answer
                          << ", Duration=" << elapsed.count() << "s\n";
            }
        }
    }

    return 0;
}
```

Таблица с результатами экспериментов

Formula: x^2

Размерность задачи	Время выполнения последовательной, с	2 потока			4 потока		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
10 000 000	0,0434175	0,0234857	1,848678132	0,9243390659	0,0167557	2,591207768	0,647801942
100 000 000	0,430051	0,206497	2,082601684	1,041300842	0,0992828	4,331576064	1,082894016
1 000 000 000	3,18256	1,61665	1,968614109	0,9843070547	0,84201	3,779717581	0,9449293951

Formula: sin(x)

Размерность задачи	Время выполнения последовательной, с	2 потока			4 потока		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
10 000 000	0,0943278	0,0492975	1,91343983	0,9567199148	0,0282469	3,339403616	0,834850904
100 000 000	0,926216	0,508808	1,82036446	0,9101822298	0,27539	3,363288427	0,8408221068
1 000 000 000	9,21183	5,07982	1,813416617	0,9067083086	2,66603	3,455261194	0,8638152984

Formula: exp(-x^2)

Размерность задачи	Время выполнения последовательной, с	2 потока			4 потока		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
10 000 000	0,0567041	0,0334473	1,695326678	0,847663339	0,0437966	1,294714658	0,3236786646
100 000 000	0,550385	0,371061	1,48327364	0,7416368198	0,197158	2,791593544	0,6978983861
1 000 000 000	5,47048	3,29514	1,660166184	0,8300830921	1,80302	3,034065069	0,7585162672