



**Министерство науки и высшего образования Российской
Федерации**
**Федеральное государственное бюджетное образовательное
учреждение высшего образования**
**«Московский государственный технический университет имени
Н.Э. Баумана**
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №4
по дисциплине "Анализ Алгоритмов"

Тема Параллельное программирование

Студент Рядинский К. В.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение	3
1 Аналитическая часть	4
1.1 Алгоритм вычисления среднего арифметического строки матрицы	4
1.2 Параллельная реализация вычисления среднего арифметического строки матрицы	4
1.3 Вывод	4
2 Конструкторская часть	6
2.1 Схемы алгоритмов	6
2.2 Описание используемых типов данных	9
2.3 Структура ПО	10
2.4 Способы тестирования и классы эквивалентности	10
2.5 Вывод	10
3 Технологическая часть	11
3.1 Средства реализации	11
3.2 Листинги кода	11
3.3 Тестирование	13
3.4 Вывод	13
4 Экспериментальная часть	14
4.1 Пример работы программы	14
4.2 Технические характеристики	14
4.3 Временные характеристики	15
4.4 Вывод	15
Заключение	16
Список литературы	17

Введение

Параллельные вычисления часто используются для увеличения скорости выполнения программ. Однако приемы, применяемые для однопоточных машин, для параллельных могут не подходить.

В данной лабораторной работе будет рассмотрено и реализовано параллельное программирование на примере задачи вычисления среднего арифметического строки матрицы.

Целью данной работы является изучения параллельных вычислений на материале задачи вычисления среднего арифметического строки матрицы.

В рамках выполнения работы необходимо решить следующие задачи:

1. Изучения основ параллельных вычислений.
2. Выбор и обоснование языка программирования, для решения данной задачи.
3. Применение изученных основ для реализации многопоточности на материале задачи вычисления среднего арифметического строки матрицы.
4. Получения практических навыков.
5. Сравнительный анализ параллельной и однопоточной реализации алгоритма вычисления среднего арифметического строки матрицы.
6. Экспериментальное подтверждение различий во временной эффективности реализации однопоточной и многопоточной версии вычисления среднего арифметического строки матрицы.
7. Описание и обоснование полученных результатов.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритм вычисления среднего арифметического строки матрицы и его параллельная реализация.

1.1 Алгоритм вычисления среднего арифметического строки матрицы

Данный алгоритм создан для вычисления среднего арифметического строки матрицы следующим образом: для каждой вычисляется ее сумма и делится на длину ¹ этой строки.

$$\bar{A}_i = \frac{\sum_{j=1}^n A_{i,j}}{n}, i = \overline{1, m} \quad (1)$$

1.2 Параллельная реализация вычисления среднего арифметического строки матрицы

Поскольку в рассмотренном выше алгоритме каждая строка обрабатывается независимо, появляется возможность распределить обработку строк среди потоков.

1.3 Вывод

В данном разделе были рассмотрены алгоритм вычисления среднего арифметического строки матрицы и его параллельная реализация.

В качестве входных данных в программу будут подаваться, размер матрицы, вещественная матрица (количество строк и столбцов). На выходе будет выдаваться массив, состоящий из средних арифметических строк матрицы. Ограничением для работы программного продукта будет являться то, что размеры матриц должны быть целыми положительными числами, а сами матрицы состоять только из вещественных значений.

¹Длиной строки мы будем называть количество элементов принадлежащей этой строке

Реализуемое программное обеспечение будет работать в пользовательском и экспериментальном режимах. В пользовательском режиме можно будет ввести матрицу и программа выведет среднее арифметическое строк данной матрицы, в экспериментальном режиме будет проводиться сравнение временных характеристик последовательного и параллельного алгоритма нахождения среднего арифметического строк матрицы при разном количестве потоков.

Критерием, по которому данная реализация будет сравниваться с другими реализациями, является время выполнения реализации алгоритма.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов (1-2) и схемы вспомогательных функций (3-4)). Описаны используемые типы данных, структура программного обеспечения (далее ПО) и классы эквивалентности.

2.1 Схемы алгоритмов



Рис. 1 — Схема последовательного алгоритма

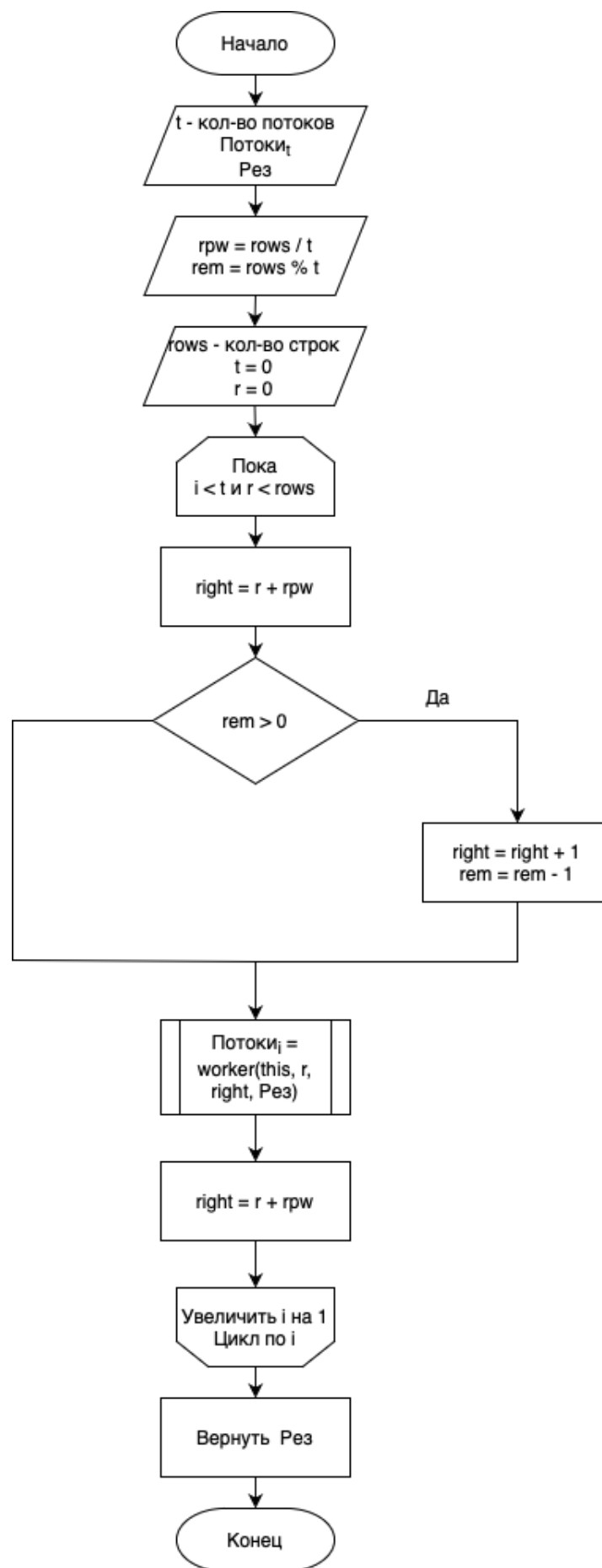


Рис. 2 — Схема параллельного алгоритма

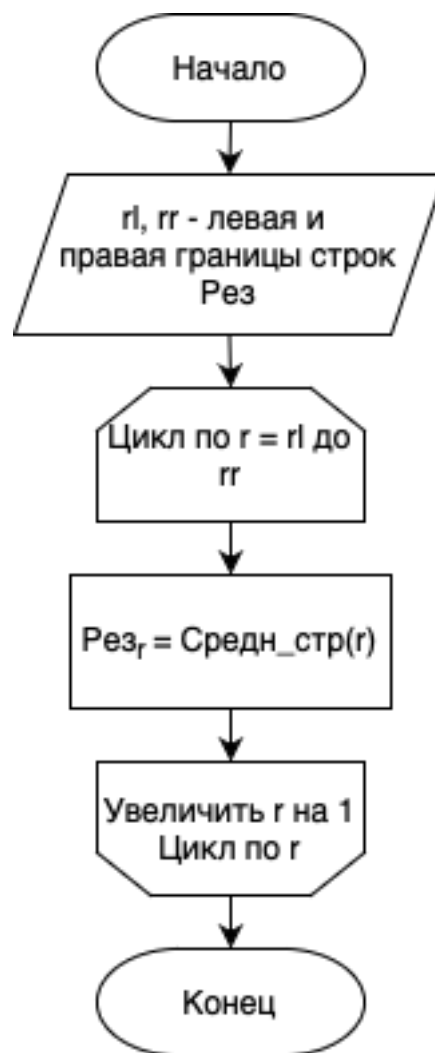


Рис. 3 — Схема вспомогательной функции `worker`

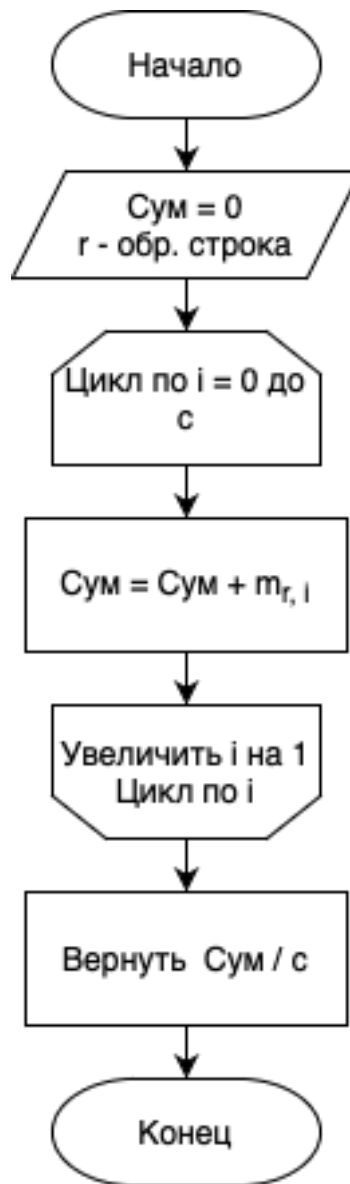


Рис. 4 — Схема вспомогательной функции вычисления среднего арифметического строки

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

1. Матрица — одномерный массив типа float.
2. Количество строк матрицы — целое число типа long long.
3. Количество столбцов матрицы — целое число типа long long.
4. Результирующий вектор — одномерный массив типа float.
5. Массив потоков — одномерный массив типа std::thread.

2.3 Структура ПО

ПО будет состоять из следующих модулей:

1. `main.cpp` — модуль, содержащий точку входа программы.
2. `matrix.cpp` — модуль, содержащий код алгоритмов нахождения среднего арифметического матрицы.

2.4 Способы тестирования и классы

эквивалентности

При тестировании алгоритмов была выбрана методика тестирования черным ящиком. Были выделены следующие классы эквивалентности:

1. Матрица, заполненная случайными положительными числами.
2. Матрица, заполненная построчно равными элементами.
3. Матрица, содержащая один элемент.
4. Пустая матрица.

2.5 Вывод

В данном разделе были рассмотрены схемы алгоритмов (1-2) и схемы вспомогательных функций (3-4)). Описаны используемые типы данных, структура ПО, способы тестирования и классы эквивалентности.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Средства реализации

К языку программирования выдвигаются следующие требования:

1. Возможность порождать системные потоки.
2. Возможность производить замер времени выполнения части программы.
3. Существуют среды разработки для этого языка.

По этим требованиям был выбран язык программирования C++.

3.2 Листинги кода

Листинг 1: Последовательный алгоритм

```
1 real row_mean(const Matrix &m, size_t r) {
2     real acc = 0;
3
4     for (size_t c = 0; c < m.getcols(); c++) {
5         acc += m(r, c);
6     }
7
8     return acc / m.getcols();
9 }
10
11 std::vector<real> Matrix::rows_mean() {
12     std::vector<real> res(rows);
13
14
15     for (size_t i = 0; i < rows; i++) {
16         res[i] = row_mean(*this, i);
17     }
18
19     return res;
20 }
```

Листинг 2: Параллельный алгоритм

```

1 static real row_mean(const real *ptr, size_t l) {
2     real acc = 0;
3
4     for (size_t i = 0; i < l; i++) {
5         acc += ptr[i];
6     }
7
8     return acc / l;
9 }
10
11 static void rows_worker(const Matrix &m, size_t rl, size_t rr
12     , std::vector<real> &res) {
13     for (size_t r = rl; r < rr; r++) {
14         res[r] = row_mean(m.getdata().data() + (r * m.getcols
15             ()), m.getcols());
16     }
17 }
18
19 std::vector<real> Matrix::rows_mean_parallel(size_t t_num) {
20     std::vector<std::thread> threads(t_num);
21     std::vector<real> res(rows);
22
23     size_t rpw = rows / t_num; // rows per thread
24     size_t remainder = rows % t_num;
25
26     size_t t = 0;
27     size_t r = 0;
28
29     while (t < t_num && r < rows) {
30         size_t right = r + rpw;
31
32         if (remainder > 0) {
33             right++;
34             remainder--;
35         }
36
37         threads[t] = std::thread(rows_worker, std::cref(*this
38             ), r, right, std::ref(res));
39     }

```

```

37         r = right;
38         t++;
39     }
40
41     for (auto && i : threads) {
42         i.join();
43     }
44
45     return res;
46 }

```

3.3 Тестирование

Тестирование проводилось методом черный ящик.

Таблица 1 — Тестирование функций нахождения среднего арифметического строк матрицы

Ввод	Фактический результат	Ожидаемый результат
1 2 3	2	2
()	()	()
0	0	0
1 1 1 2 2 2 1 2 1	1, 2, 1.333333	1, 2, 1.333333

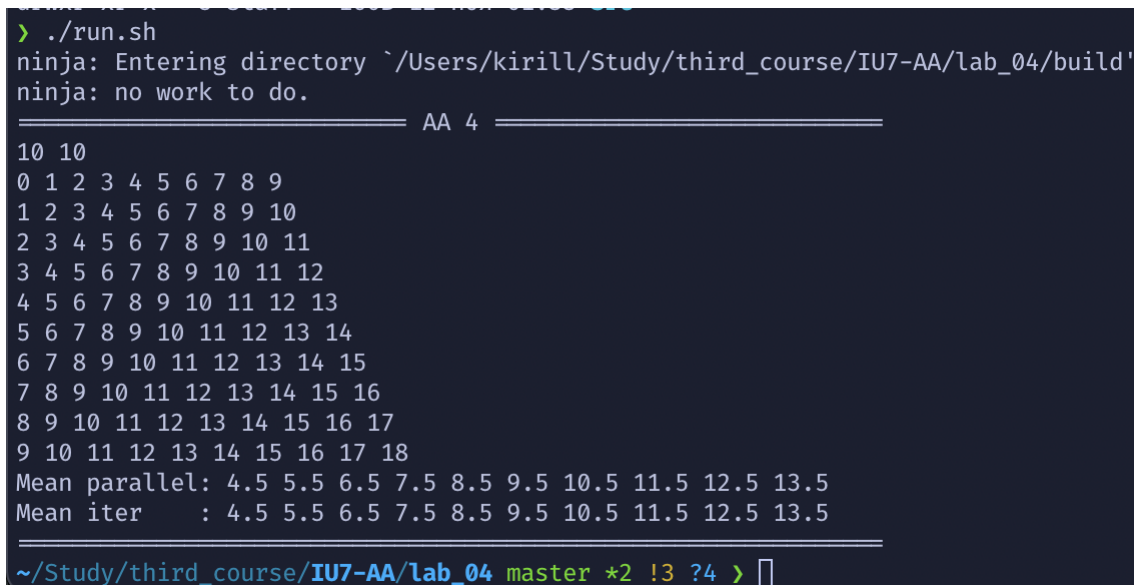
3.4 Вывод

В данном разделе были представлены средства реализации, листинги кода и проведено тестирование кода.

4 Экспериментальная часть

В данном разделе будет произведено сравнение временных характеристик вышеизложенных алгоритмов.

4.1 Пример работы программы



```
> ./run.sh
ninja: Entering directory `/Users/kirill/Study/third_course/IU7-AA/lab_04/build'
ninja: no work to do.

===== AA 4 =====
10 10
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12
4 5 6 7 8 9 10 11 12 13
5 6 7 8 9 10 11 12 13 14
6 7 8 9 10 11 12 13 14 15
7 8 9 10 11 12 13 14 15 16
8 9 10 11 12 13 14 15 16 17
9 10 11 12 13 14 15 16 17 18
Mean parallel: 4.5 5.5 6.5 7.5 8.5 9.5 10.5 11.5 12.5 13.5
Mean iter    : 4.5 5.5 6.5 7.5 8.5 9.5 10.5 11.5 12.5 13.5

~/Study/third_course/IU7-AA/lab_04 master *2 !3 ?4 > □
```

Рис. 5 — Пример работы программы

4.2 Технические характеристики

Технические характеристики электронно-вычислительной машины, на которой выполнялось тестирование:

- операционная система: macOS BigSur версия 11.4;
- оперативная память: 8 гигабайт LPDDR4;
- процессор: Apple M1.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

4.3 Временные характеристики

Замер времени выполнения функций был произведен с помощью библиотеки Google Benchmark. Эта библиотека замеряет процессорное или реальное время (по выбору пользователя) выполнения функций, усредняет это значение и выдает результат в желаемом формате (csv, json, вывод в терминал).

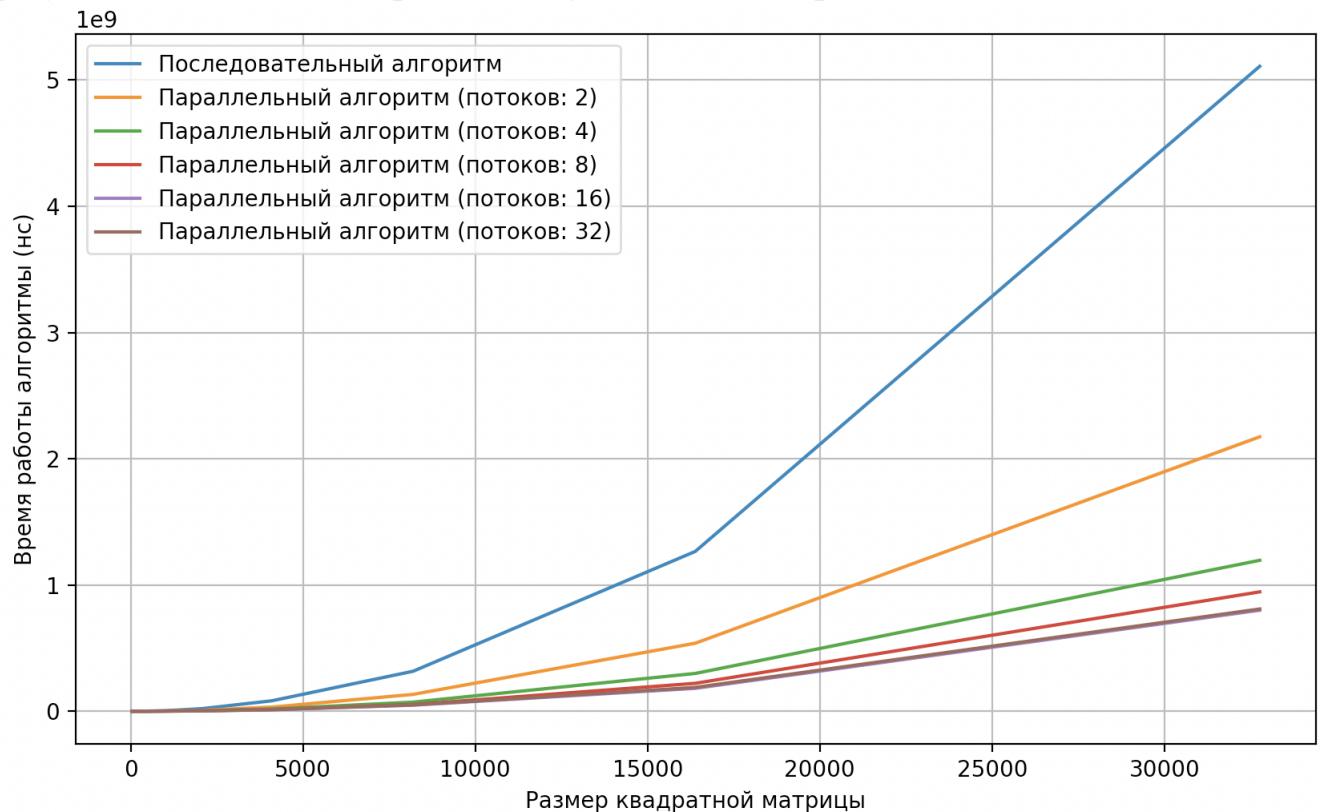


Рис. 6 — Зависимость времени выполнения алгоритма от размера квадратной матрицы

Как видно из графика 6, параллельная реализация алгоритма быстрее последовательного как минимум в 2.5 раза для двух потоков. Также из графика видно, что для машины, на которой производилось тестирование, не имеет смысла порождать более 8 потоков, так как производительность почти не растет, но на создание потоков уходят ресурсы (память).

4.4 Вывод

В данном разделе было произведено сравнение реализаций алгоритмов и зависимость времени работы от количества потоков и сделаны соответствующие заключения.

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы, которые в дальнейшем потребовались для параллельной и последовательной реализаций алгоритма вычисления среднего арифметического строки матрицы.

В рамках данной лабораторной работы была достигнута цель и выполнены следующие задачи:

1. Изучены основы параллельных вычислений.
2. Выбран и обоснован язык программирования, для решения данной задачи.
3. Применены изученные основы для реализации многопоточности на материале задачи вычисления среднего арифметического строки матрицы.
4. Получены практические навыки.
5. Был проведен сравнительный анализ параллельной и однопоточной реализации алгоритма вычисления среднего арифметического строки матрицы.
6. Было экспериментально подтверждено различия во временной эффективности реализации однопоточной и многопоточной версии вычисления среднего арифметического строки матрицы.
7. Были описаны и обоснованы полученные результаты.

Список литературы

1. Visual Studio Code [Электронный ресурс], режим доступа: <https://code.visualstudio.com/> (дата обращения: 30 ноября 2021 г.)
2. LPDDR4 [Электронный ресурс] <https://ru.wikipedia.org/wiki/LPDDR#LPDDR4> (дата обращения: 30 ноября 2021 г.)
3. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и Анализ. - Наука Физматлит, 2007. - 376.
4. Библиотека Chrono [Электронный ресурс] Режим доступа: <https://en.cppreference.com/w/cpp/chrono> (дата обращения: 30 ноября 2021 г.).
5. Библиотека для работы с потоками thread [Электронный ресурс] Режим доступа: <https://en.cppreference.com/w/cpp/thread> (дата обращения: 30 ноября 2021 г.).