

Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"
Кафедра "Вычислительная математика и программирование"

**Лабораторная работа №2 по курсу
“Операционные системы”**

Студент: Былькова Кристина Алексеевна

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 10

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1	Репозиторий	3
2	Цель работы	3
3	Задание	3
4	Описание работы программы	3
5	Исходный код	4
6	Тесты	9
7	Консоль	11
8	Запуск тестов	12
9	Выводы	13

1 Репозиторий

https://github.com/kr1st1na0/OS_labs

2 Цель работы

Приобретение практических навыков в:

- Управлении потоками в ОС
- Обеспечении синхронизации между потоками

3 Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

4 Описание работы программы

Необходимо было написать программу для решения системы линейных уравнений методом Гаусса. В данном методе можно распараллелить прямой ход: а именно нахождение максимального элемента и приведение к ступенчатому виду. Параллелить обратный ход нет смысла, так как прирост эффективности слишком мал. Я представила систему линейных уравнений в матричном виде и распределила строчки по потокам. В итоге каждый поток обрабатывал определенное количество строк, и в результате получила итоговую матрицу. Далее по алгоритму нашла искомый вектор. В ходе выполнения лабораторной работы я использовала следующие системные вызовы:

- `pthread_create()` - создание потока
- `pthread_join()` - ожидание завершения потока

5 Исходный код

lab2.hpp

```
1 #pragma once
2
3 #include <vector>
4 #include <cstdlib>
5 #include <algorithm>
6 #include <atomic>
7 #include <math.h>
8 #include <pthread.h>
9
10 using ldbl = long double;
11
12 using TVector = std::vector<ldbl>;
13 using TMatrix = std::vector<TVector>;
14
15 struct Args {
16     int startRow = 0;
17     int endRow = 0;
18     TMatrix *lhs = nullptr;
19     TVector *rhs = nullptr;
20     int leadRow = 0;
21 };
22
23 struct MaxWithRow {
24     ldbl value;
25     int row;
26 };
27
28 struct ArgsForMax {
29     int start = 0;
30     int end = 0;
31     std::vector<MaxWithRow> *maxElements = nullptr;
32     const TMatrix *matrix = nullptr;
33     long threadNum = 0;
34 };
35
36
37 void *MaxElem(void *arguments);
38 int MaxElemRowParal(const TMatrix &matrix, int start, long
    threadAmount);
39 int MaxElemRow(const TMatrix &matrix, int start);
40 void SwapRows(TMatrix &lhs, TVector &rhs, int first, int second);
41 void *Normalization(void *arguments);
42 TVector GaussMethod(long threadAmount, const TMatrix &lhs, const
    TVector &rhs);
```

lab2.cpp

```
1 #include <iostream>
2
3 #include "lab2.hpp"
4
5 void *MaxElem(void *arguments) {
6     const auto &args = *(reinterpret_cast<ArgsForMax *>(arguments)
    );
7     auto start = args.start;
8     auto end = args.end;
9     auto &maxElements = *args.maxElements;
```

```

10     auto &matrix = *args.matrix;
11     auto &threadNum = args.threadNum;
12     ldbl maxElem = fabs(matrix[start][start]);
13     int row = start;
14     for (int i = start; i < end; ++i) {
15         if (fabs(matrix[i][start]) > maxElem) {
16             maxElem = fabs(matrix[i][start]);
17             row = i;
18         }
19     }
20     if (maxElem == 0) {
21         maxElements[threadNum] = {0, -1};
22         return nullptr;
23     }
24     maxElements[threadNum] = {maxElem, row};
25     return nullptr;
26 }
27
28 int MaxElemRowParal(const TMatrix &matrix, int start, long
threadAmount) {
29     std::vector<ArgsForMax> threadArgs(threadAmount);
30     long threadAmountPerIter = std::min(threadAmount, (long)(
matrix.size() - start));
31     if (threadAmountPerIter == 0) {
32         return start;
33     }
34     long rowsPerThread = std::max(1L, (long)(((matrix.size()) -
start) / threadAmountPerIter));
35     std::vector<MaxWithRow> maxElements(threadAmountPerIter);
36     ldbl absoluteMax = fabs(matrix[start][start]);
37     int row = start;
38     std::vector<pthread_t> threads(threadAmountPerIter);
39     for (long n = 0; n < threadAmountPerIter; ++n) {
40         threadArgs[n].start = start + n * rowsPerThread;
41         threadArgs[n].end = (n == threadAmountPerIter - 1) ?
matrix.size() : (threadArgs[n].start + rowsPerThread);
42         threadArgs[n].maxElements = &maxElements;
43         threadArgs[n].matrix = &matrix;
44         threadArgs[n].threadNum = n;
45         pthread_create(&threads[n], nullptr, MaxElem,
reinterpret_cast<void *>(&threadArgs[n]));
46     }
47     for (auto &thread : threads) {
48         pthread_join(thread, nullptr);
49     }
50     for (int i = 0; i < threadAmountPerIter; ++i) {
51         if (maxElements[i].value > absoluteMax) {
52             absoluteMax = maxElements[i].value;
53             row = maxElements[i].row;
54         }
55     }
56     return row;
57 }
58
59 int MaxElemRow(const TMatrix &matrix, int start) {
60     int matrixSize = matrix.size();
61     ldbl maxElem = fabs(matrix[start][start]);
62     int row = start;
63     for (int i = start; i < matrixSize; ++i) {

```

```

64         if (fabs(matrix[i][start]) > maxElem) {
65             maxElem = fabs(matrix[i][start]);
66             row = i;
67         }
68     }
69     if (maxElem == 0) {
70         return -1;
71     }
72     return row;
73 }
74
75 void SwapRows(TMatrix &lhs, TVector &rhs, int first, int second) {
76     lhs[first].swap(lhs[second]);
77     std::swap(rhs[first], rhs[second]);
78 }
79
80 void *Normalization(void *arguments) {
81     const auto &args = *(reinterpret_cast<Args *>(arguments));
82     auto startRow = args.startRow;
83     auto endRow = args.endRow;
84     auto &leftMatrix = *args.lhs;
85     auto &rightVector = *args.rhs;
86     auto leadRow = args.leadRow;
87     int matrixSize = leftMatrix.size();
88     for (int i = startRow; i < endRow; ++i) {
89         ldbl coef = -leftMatrix[i][leadRow] / leftMatrix[leadRow][
leadRow];
90         leftMatrix[i][leadRow] = 0.0;
91         for (int j = leadRow + 1; j < matrixSize; ++j) {
92             leftMatrix[i][j] += leftMatrix[leadRow][j] * coef;
93         }
94         rightVector[i] += rightVector[leadRow] * coef;
95     }
96     return nullptr;
97 }
98
99 TVector GaussMethod(long threadAmount, const TMatrix &Mlhs, const
TVector &Vrhs) {
100     TMatrix lhs = Mlhs;
101     TVector rhs = Vrhs;
102
103     long matrixSize = lhs.size();
104     int leadRow = 0;
105     threadAmount = std::min(threadAmount, matrixSize);
106     std::vector<Args> threadArgs(threadAmount);
107     for (int i = 0; i < matrixSize; ++i) {
108         // Parallelization for max elem
109         leadRow = (threadAmount > 1) ? MaxElemRowParal(lhs,
leadRow, threadAmount) : MaxElemRow(lhs, leadRow);
110         if (leadRow == -1) {
111             std::cout << "Unable to get the solution due to zero
column" << std::endl;
112             return {0};
113         }
114         // Leading string conversion
115         ldbl leadElem = lhs[leadRow][i];
116         for (int k = 0; k < matrixSize; ++k) {
117             lhs[leadRow][k] /= leadElem;
118         }

```

```

119         rhs[leadRow] /= leadElem;
120         // Swap rows
121         if (leadRow != i) {
122             SwapRows(lhs, rhs, i, leadRow);
123         } else {
124             ++leadRow;
125         }
126         // Parallelization for strings
127         if (threadAmount > 1) {
128             if (i != matrixSize - 1) {
129                 long threadAmountPerIter = std::min(threadAmount,
matrixSize - i) - 1;
130                 long rowsPerThread = std::max(1L, (matrixSize - 1
- i) / threadAmountPerIter);
131                 std::vector<pthread_t> threads(threadAmountPerIter
);
132                 for (int n = 0; n < threadAmountPerIter; ++n) {
133                     threadArgs[n].startRow = i + 1 + n *
rowsPerThread;
134                     threadArgs[n].endRow = (n ==
threadAmountPerIter - 1) ? matrixSize : (threadArgs[n].startRow
+ rowsPerThread);
135                     threadArgs[n].lhs = &lhs;
136                     threadArgs[n].rhs = &rhs;
137                     threadArgs[n].leadRow = i;
138                     pthread_create(&threads[n], nullptr,
Normalization, reinterpret_cast<void *>(&threadArgs[n]));
139                 }
140                 for (auto &thread : threads) {
141                     pthread_join(thread, nullptr);
142                 }
143             }
144         } else {
145             threadArgs[0].startRow = i + 1;
146             threadArgs[0].endRow = matrixSize;
147             threadArgs[0].lhs = &lhs;
148             threadArgs[0].rhs = &rhs;
149             threadArgs[0].leadRow = i;
150             Normalization(&threadArgs[0]);
151         }
152     }
153     // Reverse move
154     TVector answer(matrixSize);
155     for (int k = matrixSize - 1; k >= 0; --k) {
156         answer[k] = rhs[k];
157         for (int i = 0; i < k; ++i) {
158             rhs[i] = rhs[i] - lhs[i][k] * answer[k];
159         }
160     }
161     return answer;
162 }

```

main.cpp

```

1 #include <iostream>
2
3 #include "lab2.hpp"
4
5 int main(int argc, char *argv[]) {
6     if (argc != 2) {

```

```

7         std::cout << "Not enough arguments" << std::endl;
8         return -1;
9     }
10
11     long threadAmount = std::atol(argv[1]);
12
13     int n;
14     std::cin >> n;
15
16     TMatrix lhs(n, TVector(n));
17     TVector rhs(n);
18
19     auto readMatrix = [n](TMatrix &matrix) {
20         for (int i = 0; i < n; ++i) {
21             for(int j = 0; j < n; ++j) {
22                 std::cin >> matrix[i][j];
23             }
24         }
25     };
26
27     auto readVector = [n](TVector &vector) {
28         for (int i = 0; i < n; ++i) {
29             std::cin >> vector[i];
30         }
31     };
32
33     auto printMatrix = [n](TMatrix &matrix, TVector &vector) {
34         for (int i = 0; i < n; ++i) {
35             std::cout << "| ";
36             for (int j = 0; j < n; ++j) {
37                 std::cout << matrix[i][j] << ' ';
38             }
39             std::cout << '|';
40             std::cout << " | x" << i + 1 << " | | " << vector[i]
41             << " |" << std::endl;
42         }
43     };
44
45     readMatrix(lhs);
46     readVector(rhs);
47     std::cout << "The system of equations in matrix:" << std::endl
48     ;
49     printMatrix(lhs, rhs);
50
51     auto answer = GaussMethod(threadAmount, lhs, rhs);
52     std::cout << "The solution:" << std::endl;
53     for (int i = 0; i < n; ++i) {
54         std::cout << "x" << i + 1 << " = " << answer[i] << std::
55         endl;
56     }
57
58     return 0;
59 }

```


6 Тесты

```
1 #include <gtest/gtest.h>
2
3 #include "lab2.hpp"
4
5 #include <limits>
6 #include <chrono>
7
8 const ldbl  LDBL_PRECISION = 0.0001;
9
10 namespace {
11     TMatrix GenerateMatrix(int n) {
12         TMatrix result(n, TVector(n));
13         std::srand(std::time(nullptr));
14         for(int i = 0; i < n; ++i) {
15             for(int j = 0; j < n; ++j) {
16                 result[i][j] = std::rand() % 100;
17             }
18         }
19         return result;
20     }
21
22     TVector GenerateVector(int n) {
23         TVector result(n);
24         std::srand(std::time(nullptr));
25         for(int i = 0; i < n; ++i) {
26             result[i] = std::rand() % 100;
27         }
28         return result;
29     }
30 }
31
32 bool AreEqual(const TVector &first, const TVector &second) {
33     if (first.size() != second.size()) {
34         return false;
35     }
36     for(size_t i = 0; i < first.size(); ++i) {
37         if (!(std::fabs(first[i] - second[i]) < LDBL_PRECISION)) {
38             return false;
39         }
40     }
41     return true;
42 }
43
44 TEST(SecondLabTests, SingleThreadYieldsCorrectResults) {
45     ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{1, -1}, {2, 1}},
46     TVector{-5, -7}), TVector{-4, 1}) );
47
48     ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{2, 4, 1}, {5, 2,
49     1}, {2, 3, 4}}, TVector{36, 47, 37}), TVector{7, 5, 2}) );
50
51     ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{3, 2, -5}, {2,
52     -1, 3}, {1, 2, -1}}, TVector{-1, 13, 9}), TVector{3, 5, 4}) );
53
54     ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{1, 1, 2, 3}, {1,
55     2, 3, -1}, {3, -1, -1, -2}, {2, 3, -1, -1}}, TVector{1, -4,
56     -4, -6}),
57     TVector{-1, -1, 0, 1}) );
58 }
```

```

53 }
54
55 TEST(SecondLabTest, ThreadConfigurations) {
56     auto performTestForGivenSize = [](int n, int maxThreadCount) {
57         auto m = GenerateMatrix(n);
58         auto v = GenerateVector(n);
59         auto result = GaussMethod(1, m, v);
60
61         for(int i = 2; i < maxThreadCount; ++i) {
62             ASSERT_TRUE( AreEqual(GaussMethod(i, m, v), result) );
63         }
64     };
65
66     performTestForGivenSize(3, 10);
67     performTestForGivenSize(10, 10);
68     performTestForGivenSize(100, 15);
69     performTestForGivenSize(1000, 4);
70 }
71
72 TEST(SecondLabTest, PerformanceTest) {
73     auto getAvgTime = [](int threadCount) {
74         auto m = GenerateMatrix(3000);
75         auto v = GenerateVector(3000);
76
77         constexpr int runsCount = 1;
78
79         double avg = 0;
80
81         for(int i = 0; i < runsCount; ++i) {
82             auto begin = std::chrono::high_resolution_clock::now()
;
83             GaussMethod(threadCount, m, v);
84             auto end = std::chrono::high_resolution_clock::now();
85             avg += std::chrono::duration_cast<std::chrono::
milliseconds>(end - begin).count();
86         }
87
88         return avg / runsCount;
89     };
90
91     auto singleThread = getAvgTime(1);
92     auto multiThread = getAvgTime(4);
93
94     std::cout << "Avg time for 1 thread: " << singleThread << '\n'
;
95     std::cout << "Avg time for 4 threads: " << multiThread << '\n'
;
96
97     EXPECT_GE(singleThread, multiThread);
98 }
99
100 int main(int argc, char *argv[]) {
101     testing::InitGoogleTest(&argc, argv);
102     std::cout.precision(15);
103     std::cout.setf(std::ios_base::fixed, std::ios_base::floatfield
);
104     return RUN_ALL_TESTS();
105 }

```

7 Консоль

```
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab2$ ./lab2 10
2
1 -1
2 1
-5 -7
The system of equations in matrix:
| 1 -1 | | x1 | | -5 |
| 2 1 | | x2 | | -7 |
The solution:
x1 = -4
x2 = 1
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab2$ ./lab2 8
3
2 4 1
5 2 1
2 3 4
36 47 37
The system of equations in matrix:
| 2 4 1 | | x1 | | 36 |
| 5 2 1 | | x2 | | 47 |
| 2 3 4 | | x3 | | 37 |
The solution:
x1 = 7
x2 = 5
x3 = 2
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab2$ ./lab2 1
4
1 1 2 3
1 2 3 -1
3 -1 -1 -2
2 3 -1 -1
1 -4 -4 -6
The system of equations in matrix:
| 1 1 2 3 | | x1 | | 1 |
| 1 2 3 -1 | | x2 | | -4 |
| 3 -1 -1 -2 | | x3 | | -4 |
| 2 3 -1 -1 | | x4 | | -6 |
The solution:
x1 = -1
x2 = -1
x3 = 0
x4 = 1
```

8 Запуск тестов

```
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/tests$ ./lab2_test
[=====] Running 3 tests from 2 test suites.
[-----] Global test environment set-up.
[-----] 1 test from SecondLabTests
[ RUN      ] SecondLabTests.SingleThreadYieldsCorrectResults
[          OK ] SecondLabTests.SingleThreadYieldsCorrectResults (0 ms)
[-----] 1 test from SecondLabTests (0 ms total)

[-----] 2 tests from SecondLabTest
[ RUN      ] SecondLabTest.ThreadConfigurations
[          OK ] SecondLabTest.ThreadConfigurations (10492 ms)
[ RUN      ] SecondLabTest.PerformanceTest
Avg time for 1 thread: 105372.0000000000000000
Avg time for 4 threads: 40388.0000000000000000
[          OK ] SecondLabTest.PerformanceTest (146061 ms)
[-----] 2 tests from SecondLabTest (156553 ms total)

[-----] Global test environment tear-down
[=====] 3 tests from 2 test suites ran. (156553 ms total)
[ PASSED   ] 3 tests.
```

Из тестов видно, что при запуске с большим количеством потоков, программа работает быстрее, а на результат это никак не влияет. Сравнение было произведено на 1 и 4 потоках.

Ускорение:

$$S_4 = \frac{T_1}{T_4} < 4 \quad (1)$$

$$S_4 = \frac{105372}{40388} = 2.6 < 4 \quad (2)$$

Эффективность:

$$X_4 = \frac{S_4}{4} < 1 \quad (3)$$

$$X_4 = \frac{4}{40388} = 0.000099 < 1 \quad (4)$$

9 Выводы

В результате выполнения данной лабораторной работы была написана программа на языке C++ для решения системы линейных уравнений методом Гаусса, обрабатывающая данные в многопоточном режиме. Я приобрела практические навыки в управлении потоками в ОС и обеспечении синхронизации между потоками.