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

```
#pragma once

#include <vector>
#include <cstdlib>
#include <algorithm>
#include <atomic>
#include <math.h>
#include <pthread.h>

using ldbl = long double;

using TVector = std::vector<ldbl>;
using TMatrix = std::vector<TVector>;

struct Args {
    int startRow = 0;
    int endRow = 0;
    TMatrix *lhs = nullptr;
    TVector *rhs = nullptr;
    int leadRow = 0;
};

struct MaxWithRow {
    ldbl value;
    int row;
};

struct ArgsForMax {
    int start = 0;
    int end = 0;
    std::vector<MaxWithRow> *maxElements = nullptr;
    const TMatrix *matrix = nullptr;
    long threadNum = 0;
};


void *MaxElem(void *arguments);
int MaxElemRowParal(const TMatrix &matrix, int start, long
    threadAmount);
int MaxElemRow(const TMatrix &matrix, int start);
void SwapRows(TMatrix &lhs, TVector &rhs, int first, int second);
void *Normalization(void *arguments);
TVector GaussMethod(long threadAmount, const TMatrix &lhs, const
    TVector &rhs);
```

lab2.cpp

```
#include <iostream>

#include "lab2.hpp"

void *MaxElem(void *arguments) {
    const auto &args = *(reinterpret_cast<ArgsForMax *>(arguments)
    );
    auto start = args.start;
    auto end = args.end;
    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]
    << " |" << std::endl;
41          }
42      };
43
44      readMatrix(lhs);
45      readVector(rhs);
46      std::cout << "The system of equations in matrix:" << std::endl
    ;
47      printMatrix(lhs, rhs);
48
49      auto answer = GaussMethod(threadAmount, lhs, rhs);
50      std::cout << "The solution:" << std::endl;
51      for (int i = 0; i < n; ++i) {
52          std::cout << "x" << i + 1 << " = " << answer[i] << std::
    endl;
53      }
54
55      return 0;
56  }
```

6 Тесты

```cpp
#include <gtest/gtest.h>

#include "lab2.hpp"

#include <limits>
#include <chrono>

const ldbl LDBL_PRECISION = 0.0001;

namespace {
    TMatrix GenerateMatrix(int n) {
        TMatrix result(n, TVector(n));
        std::srand(std::time(nullptr));
        for(int i = 0; i < n; ++i) {
            for(int j = 0; j < n; ++j) {
                result[i][j] = std::rand() % 100;
            }
        }
        return result;
    }

    TVector GenerateVector(int n) {
        TVector result(n);
        std::srand(std::time(nullptr));
        for(int i = 0; i < n; ++i) {
            result[i] = std::rand() % 100;
        }
        return result;
    }
}

bool AreEqual(const TVector &first, const TVector &second) {
    if (first.size() != second.size()) {
        return false;
    }
    for(size_t i = 0; i < first.size(); ++i) {
        if (!(std::fabs(first[i] - second[i]) < LDBL_PRECISION)) {
            return false;
        }
    }
    return true;
}

TEST(SecondLabTests, SingleThreadYieldsCorrectResults) {
    ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{1, -1}, {2, 1}},
    TVector{-5, -7}), TVector{-4, 1}) );

    ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{2, 4, 1}, {5, 2,
    1}, {2, 3, 4}}, TVector{36, 47, 37}), TVector{7, 5, 2}) );

    ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{3, 2, -5}, {2,
   -1, 3}, {1, 2, -1}}, TVector{-1, 13, 9}), TVector{3, 5, 4}) );

    ASSERT_TRUE( AreEqual(GaussMethod(1, TMatrix{{1, 1, 2, 3}, {1,
    2, 3, -1}, {3, -1, -1, -2}, {2, 3, -1, -1}}, TVector{1, -4,
   -4, -6}),
    TVector{-1, -1, 0, 1}) );
```

```
}

TEST(SecondLabTest, ThreadConfigurations) {
    auto performTestForGivenSize = [](int n, int maxThreadCount) {
        auto m = GenerateMatrix(n);
        auto v = GenerateVector(n);
        auto result = GaussMethod(1, m, v);

        for(int i = 2; i < maxThreadCount; ++i) {
            ASSERT_TRUE( AreEqual(GaussMethod(i, m, v), result) );
        }
    };

    performTestForGivenSize(3, 10);
    performTestForGivenSize(10, 10);
    performTestForGivenSize(100, 15);
    performTestForGivenSize(1000, 4);
}

TEST(SecondLabTest, PerfomanceTest) {
    auto getAvgTime = [](int threadCount) {
        auto m = GenerateMatrix(3000);
        auto v = GenerateVector(3000);

        constexpr int runsCount = 1;

        double avg = 0;

        for(int i = 0; i < runsCount; ++i) {
            auto begin = std::chrono::high_resolution_clock::now()
    ;
            GaussMethod(threadCount, m, v);
            auto end = std::chrono::high_resolution_clock::now();
            avg += std::chrono::duration_cast<std::chrono::
    milliseconds>(end - begin).count();
        }

        return avg / runsCount;
    };

    auto singleThread = getAvgTime(1);
    auto multiThread = getAvgTime(4);

    std::cout << "Avg time for 1 thread: " << singleThread << '\n'
    ;
    std::cout << "Avg time for 4 threads: " << multiThread << '\n'
    ;

    EXPECT_GE(singleThread, multiThread);
}

int main(int argc, char *argv[]) {
    testing::InitGoogleTest(&argc, argv);
    std::cout.precision(15);
    std::cout.setf(std::ios_base::fixed, std::ios_base::floatfield
    );
    return RUN_ALL_TESTS();
}
```

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.PerfomanceTest
Avg time for 1 thread: 105372.000000000000000
Avg time for 4 threads: 40388.000000000000000
[       OK ] SecondLabTest.PerfomanceTest (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:

$$S_4 = \frac{T_1}{T_4} < 4 \tag{1}$$

$$S_4 = \frac{105372}{40388} = 2.6 < 4 \tag{2}$$

Эффективность:

$$X_4 = \frac{S_4}{4} < 1 \tag{3}$$

$$X_4 = \frac{4}{40388} = 0.000099 < 1 \tag{4}$$

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.