Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Операционные системы»

Студент: Ширяев Никита Алексеевич
Группа: М8О-208Б-22
Вариант: 9
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Содержание

Репозиторий

https://github.com/resdt/os-labs

Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Общие сведения о программе

Вариант: Рассчитать детерминант матрицы (используя определение детерминанта).

Общий метод и алгоритм решения

Для решения используется алгоритм, основанный на определении детерминанта матицы через перестановки:

$$\det A = \sum_{\alpha_1,\alpha_2,\ldots,\alpha_n} (-1)^{N(\alpha_1,\alpha_2,\ldots,\alpha_n)} \cdot a_{1\alpha_1} a_{2\alpha_2} \ldots a_{n\alpha_n},$$

В данном методе можно распараллеливать расчет каждого слагаемого суммы.

Исходный код

lab2.hpp

```
#pragma once

#include <iostream>
#include <vector>
#include <algorithm>
#include <pthread.h>
#include <atomic>

struct Args {
    size_t ind = 0;
    size_t permsPerThread = 0;
    std::atomic<size_t> * sum = nullptr;
    const std::vector<std::vector<int> >* matrix = nullptr;
    const std::vector<std::vector<int> >* permutations =
nullptr;

};

int calculatePermSign(const std::vector<int>& indices);
void *task(void *input);
int calculateDeterminant(size_t threadQuantity, const
std::vector<std::vector<int>>& matrix);
```

lab2.cpp

```
#include <lab2.hpp>

size_t fact(size_t n) {
    if (n == 0) {
        return 1;
    }
    return n * fact(n - 1);
}

int calculatePermSign(const std::vector<int>& indices) {
    int numInversions = 0;
    int size = indices.size();
```

```
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (indices[i] > indices[j]) {
                numInversions++;
            }
        }
    }
    return numInversions % 2 == 0 ? 1 : -1;
}

void *task(void *input) {
    const auto &args = *(reinterpret_cast<Args *>(input));
    const std::vector<std::vector<int> >& perms =
*(args.permutations);
    const std::vector<std::vector<int> >& mat =
*(args.matrix);
    size_t partOfSum = 0;
    size_t startInd = args.ind * args.permsPerThread;
    size_t endInd = startInd + args.permsPerThread;

    for (size_t i = startInd; i < endInd; i++) {
        std::vector<int> indices = perms[i];
        int permSign = calculatePermSign(indices);
        int permSummand = 1;

        for(size_t j = 0; j < indices.size(); j++) {
            permSummand *= mat[j][indices[j]];
        }
        partOfSum += (permSign * permSummand);
    }
    (*args.sum).fetch_add(partOfSum);
    return nullptr;
}

int calculateDeterminant(size_t threadQuantity, const
std::vector<std::vector<int>>& matrix) {
    size_t n = matrix.size();
```

```
    std::vector<int> indices(n);

    size_t number_of_perm = fact(n);
    if (number_of_perm % threadQuantity != 0) {
        perror("Incorrect number of threads. It must be
integers divided by the factorial of the order of the
matrix.");
        exit(EXIT_FAILURE);
    }

    std::vector<std::vector<int> >
permutations(number_of_perm);
    int determinant{0};
    for(size_t i = 0; i < n; i++) {
        indices[i] = i;
    }
    for(size_t i = 0; i < permutations.size(); i++) {
        std::next_permutation(indices.begin(),
indices.end());
        permutations[i] = indices;
    }
    if (threadQuantity > 1) {
        std::atomic<size_t> sum {0};
        const size_t actualThreadQuantity =
std::min(threadQuantity, permutations.size());
        std::vector<pthread_t> threads(actualThreadQuantity);

        const size_t permsPerThread = permutations.size() /
actualThreadQuantity;

        std::vector<Args>
argsForThread(actualThreadQuantity);

        for (size_t i = 0; i < actualThreadQuantity; i++) {
            argsForThread[i].permsPerThread = permsPerThread;
            argsForThread[i].sum = &sum;
            argsForThread[i].matrix = &matrix;
            argsForThread[i].permutations = &permutations;
```

```
            argsForThread[i].sum = &sum;
            argsForThread[i].ind = i;

            pthread_create(&threads[i], nullptr, task,
reinterpret_cast<void *>(&argsForThread[i]));
        }
        for (auto &thread : threads) {
            pthread_join(thread, nullptr);
        }
        determinant = (int)sum;
    } else {
        for (size_t i = 0; i < permutations.size(); i++) {
            int permSign =
calculatePermSign(permutations[i]);
            int permSummand = 1;
            for(size_t j = 0; j < n; j++) {
                permSummand *= matrix[j][permutations[i][j]];
            }
            determinant += (permSign * permSummand);
        }
    }
    return determinant;
}
```

lab2_test.cpp

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

#include "lab2.hpp"

TEST(secondLabTests, ONE) {
    std::vector<std::vector<int>> matrix = {
        { 1, 0, 0},
        { 0, 1, 0},
        { 0, 0, 1}
    };
    size_t threadQuantity = 1;
```

```
    ASSERT_EQ(calculateDeterminant(threadQuantity, matrix),
1);
}

TEST(secondLabTests, TWO) {
    std::vector<std::vector<int>> matrix = {
            { 3, 6, 2, 9, 2 },
            { 5, 8, 3, 5, 2 },
            { 5, 4, 8, 10, 3 },
            { 1, 4, 7, 5, 5 },
            { 6, 9, 3, 1, 4 }
    };
    size_t threadQuantity = 1;
    ASSERT_EQ(calculateDeterminant(threadQuantity, matrix),
2717);
}

TEST(secondLabTests, THREE) {
    std::vector<std::vector<int>> matrix = {
            { 3, 6, 2, 9, 2 },
            { 5, 8, 3, 5, 2 },
            { 5, 4, 8, 10, 3 },
            { 1, 4, 7, 5, 5 },
            { 6, 9, 3, 1, 4 }
    };
    size_t threadQuantity = 10;
    ASSERT_EQ(calculateDeterminant(threadQuantity, matrix),
2717);
}

TEST(secondLabTests, FOUR) {
    std::vector<std::vector<int>> matrix = {
            { 2, 5, 6, 10, 3, 1, 5, 3, 1, 5, },
            { 3, 5, 7, 10, 4, 5, 3, 7, 10, 8, },
            { 7, 5, 8, 3, 7, 1, 8, 3, 1, 7, },
            { 10, 2, 4, 5, 4, 6, 5, 8, 8, 6, },
            { 3, 2, 10, 9, 3, 5, 5, 8, 4, 6, },
            { 5, 10, 10, 2, 2, 8, 4, 1, 2, 4, },
```

```
            { 9, 1, 8, 2, 5, 1, 7, 10, 10, 6, },
            { 7, 2, 9, 8, 2, 2, 3, 8, 9, 6, },
            { 3, 3, 7, 5, 6, 10, 2, 9, 10, 4, },
            { 3, 10, 6, 10, 2, 1, 2, 10, 2, 1, }
    };
    size_t threadQuantity = 1;
    ASSERT_EQ(calculateDeterminant(threadQuantity, matrix),
294535103);
}

TEST(secondLabTests, FIVE) {
    std::vector<std::vector<int>> matrix = {
            { 2, 5, 6, 10, 3, 1, 5, 3, 1, 5, },
            { 3, 5, 7, 10, 4, 5, 3, 7, 10, 8, },
            { 7, 5, 8, 3, 7, 1, 8, 3, 1, 7, },
            { 10, 2, 4, 5, 4, 6, 5, 8, 8, 6, },
            { 3, 2, 10, 9, 3, 5, 5, 8, 4, 6, },
            { 5, 10, 10, 2, 2, 8, 4, 1, 2, 4, },
            { 9, 1, 8, 2, 5, 1, 7, 10, 10, 6, },
            { 7, 2, 9, 8, 2, 2, 3, 8, 9, 6, },
            { 3, 3, 7, 5, 6, 10, 2, 9, 10, 4, },
            { 3, 10, 6, 10, 2, 1, 2, 10, 2, 1, }
    };
    size_t threadQuantity = 10;
    ASSERT_EQ(calculateDeterminant(threadQuantity, matrix),
294535103);
}

TEST(secondLabTests, SIX) {
    std::vector<std::vector<int>> matrix = {
            { 2, 5, 6, 10, 3, 1, 5, 3, 1, 5, },
            { 3, 5, 7, 10, 4, 5, 3, 7, 10, 8, },
            { 7, 5, 8, 3, 7, 1, 8, 3, 1, 7, },
            { 10, 2, 4, 5, 4, 6, 5, 8, 8, 6, },
            { 3, 2, 10, 9, 3, 5, 5, 8, 4, 6, },
            { 5, 10, 10, 2, 2, 8, 4, 1, 2, 4, },
            { 9, 1, 8, 2, 5, 1, 7, 10, 10, 6, },
            { 7, 2, 9, 8, 2, 2, 3, 8, 9, 6, },
```

```
            { 3, 3, 7, 5, 6, 10, 2, 9, 10, 4, },
            { 3, 10, 6, 10, 2, 1, 2, 10, 2, 1, }
    };
    size_t threadQuantity = 350;
    ASSERT_EQ(calculateDeterminant(threadQuantity, matrix),
294535103);
}

int main(int argc, char *argv[]) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

main.cpp

```
#include <lab2.hpp>

int main(int argc, char const *argv[]) {
    if (argc != 2) {
        std::cout << "wrong arguments" << std::endl;
        return -1;
    }
    size_t threadQuantity = std::atol(argv[1]);
    std::vector<std::vector<int>> matrix = {
            { 3, 6, 2, 9, 2 },
            { 5, 8, 3, 5, 2 },
            { 5, 4, 8, 10, 3 },
            { 1, 4, 7, 5, 5 },
            { 6, 9, 3, 1, 4 }
    };

    int determinant = calculateDeterminant(threadQuantity,
matrix);

    std::cout << determinant << std::endl;
    return 0;
}
```

Демонстрация работы программы

```
hacker@warmachine:~/prog/my_os_labs/build/lab2$ ./lab2 5
2717
hacker@warmachine:~/prog/my_os_labs/build/lab2$
../tests/lab2_test
[==========] Running 6 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 6 tests from secondLabTests
[ RUN ] secondLabTests.ONE
[ OK ] secondLabTests.ONE (0 ms)
[ RUN ] secondLabTests.FOUR
[ OK ] secondLabTests.FOUR (3517 ms)
[ RUN ] secondLabTests.THREE
[ OK ] secondLabTests.THREE (1 ms)
[ RUN ] secondLabTests.TWO
[ OK ] secondLabTests.TWO (0 ms)
[ RUN ] secondLabTests.FIVE
[ OK ] secondLabTests.FIVE (2062 ms)
[ RUN ] secondLabTests.SIX
[ OK ] secondLabTests.SIX (2016 ms)
[----------] 6 tests from secondLabTests (7598 ms total)
[----------] Global test environment tear-down
[==========] 6 tests from 1 test suite ran. (7598 ms total)
[ PASSED ] 6 tests.
hacker@warmachine:~/prog/my_os_labs/build/lab2$
```

Выводы

В результате выполнения данной лабораторной работы была написана программа на языке C++ для нахождения детерминанта матрицы, обрабатывающая данные в многопоточном режиме. Я приобрел практические навыки в управлении потоками в ОС и обеспечении синхронизации между потоками.