**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и вычислительной техники

Кафедра ИиСП

**Лабораторная работа**

**по дисциплине «Параллельное программирование»**

Выполнил:

студент группы ПС-31

факультета Информатики

и Вычислительной Техники

специальности «Программная инженерия»

Илдаркин А. A.

Проверил:

Филимонов А. A.

г. Йошкар-Ола

2016

# Постановка задачи

Найти алгебраическое дополнение для каждого элемента матрицы.

Входные данные: произвольная матрица А размерности n х n. Заполнение начальной матрицы реализовать в главном потоке приложения.

1. Реализовать последовательный вариант программы для указанного варианта.
2. Реализовать параллельный вариант программы. Количество потоков выполнения должно являться входным параметром задачи.
3. После завершения программа должна выдавать время своей работы. Подобрать размеры матриц таким образом, чтобы время работы последовательного варианта составляло не менее одной секунды.
4. Посчитать параметры качества вашей параллельной программы и построить в Excel графики для количества потоков = [1;16] и количестве задействованных ядер = [1;4] (используйте функцию SetAffinityMask).
5. Время выполнения
6. Ускорение
7. Эффективность распараллеливания.

# Листинг кода

main.cpp:

#include "stdafx.h"

#include "ConsistendMatrixCalculator.h"

#include "MultithreadedMatrixCalculator.h"

#include "Matrix\_res.h"

int main(int argc, char \* argv[])

{

size\_t threadsCount = std::atoi(argv[1]);

CMatrix matrix(MATRIX);

auto startTime = std::chrono::steady\_clock::now();

if (threadsCount > 0)

{

CMultithreadedMatrixCalculator calc(matrix, threadsCount);

calc.CalculateMatrixAlgebraicAdds();

}

else

{

CConsistendMatrixCalculator calc(matrix);

calc.CalculateMatrixAlgebraicAdds();

}

auto endTime = std::chrono::steady\_clock::now();

std::chrono::duration<double> diff = endTime - startTime;

std::cout << diff.count() << std::endl;

return 0;

}

IMatrixCalculator.h:

#pragma once

#include "Definitions.h"

class IMatrixCalculator

{

public:

virtual ~IMatrixCalculator() = default;

virtual const Matrix & CalculateMatrixAlgebraicAdds() = 0;

};

ConsistendMatrixCalculator.h:

#pragma once

#include "IMatrixCalculator.h"

#include "Matrix.h"

class CConsistendMatrixCalculator

: public IMatrixCalculator

{

public:

CConsistendMatrixCalculator(const CMatrix & matrix);

const Matrix & CalculateMatrixAlgebraicAdds() override;

private:

CMatrix m\_matrix;

};

ConsistendMatrixCalculator.cpp:

#include "stdafx.h"

#include "ConsistendMatrixCalculator.h"

CConsistendMatrixCalculator::CConsistendMatrixCalculator(const CMatrix & matrix)

: m\_matrix(matrix)

{

}

const Matrix & CConsistendMatrixCalculator::CalculateMatrixAlgebraicAdds()

{

auto matrixSize = m\_matrix.GetInputMatrixSize();

for (size\_t i = 0; i != matrixSize.first; i++)

{

for (size\_t j = 0; j != matrixSize.second; j++)

{

m\_matrix.CalculateAlgebraicAddition(i, j);

}

}

return m\_matrix.GetOutputMatrix();

}

MultithreadedMatrixCalculator.h:

#pragma once

#include "IMatrixCalculator.h"

#include "Matrix.h"

class CMultithreadedMatrixCalculator

: public IMatrixCalculator

{

public:

CMultithreadedMatrixCalculator(const CMatrix & matrix, size\_t threadsCount);

const Matrix & CalculateMatrixAlgebraicAdds()override;

DWORD WINAPI CalculateLine(PVOID pvParam);

private:

void FillThreadsChargeMap();

private:

ThreadsChargeMap m\_threadsChargeMap;

CMatrix m\_matrix;

size\_t m\_threadsCount;

};

MultithreadedMatrixCalculator.cpp:

#include "stdafx.h"

#include "MultithreadedMatrixCalculator.h"

namespace

{

DWORD WINAPI StartMultithreadedCalculator(PVOID pvParam)

{

auto data = static\_cast<std::pair<CMultithreadedMatrixCalculator\*, int>\*>(pvParam);

return data->first->CalculateLine((PVOID)&data->second);

}

}

CMultithreadedMatrixCalculator::CMultithreadedMatrixCalculator(const CMatrix & matrix, size\_t threadsCount)

: m\_matrix(matrix)

, m\_threadsCount(threadsCount)

{

FillThreadsChargeMap();

}

const Matrix & CMultithreadedMatrixCalculator::CalculateMatrixAlgebraicAdds()

{

HANDLE \*hThreads = \*std::make\_shared<HANDLE\*>(new HANDLE[m\_threadsCount]);

DWORD \*dwThreadsId = \*std::make\_shared<DWORD\*>(new DWORD[m\_threadsCount]);

for (size\_t i = 1; i <= m\_threadsCount; i++)

{

hThreads[i - 1] = CreateThread(NULL, 0, StartMultithreadedCalculator, (PVOID)&(std::make\_pair(this, i)), 0, &dwThreadsId[i - 1]);

}

WaitForMultipleObjects(m\_threadsCount, hThreads, TRUE, INFINITE);

return m\_matrix.GetOutputMatrix();

}

void CMultithreadedMatrixCalculator::FillThreadsChargeMap()

{

auto const & matrixSize = m\_matrix.GetInputMatrixSize();

for (size\_t i = 1; i <= m\_threadsCount; i++)

{

m\_threadsChargeMap.emplace(i, std::vector<size\_t>());

}

for (size\_t i = 0, threadNumber = 1; i != matrixSize.first; i++, threadNumber++)

{

if (threadNumber > m\_threadsCount)

{

threadNumber = 1;

}

m\_threadsChargeMap.at(threadNumber).push\_back(i);

}

}

DWORD CMultithreadedMatrixCalculator::CalculateLine(PVOID pvParam)

{

auto threadNumber = \*static\_cast<int\*>(pvParam);

auto linesNumbers = m\_threadsChargeMap.at(threadNumber);

for (auto it = linesNumbers.begin(); it != linesNumbers.end(); ++it)

{

for (size\_t j = 0; j != m\_matrix.GetInputMatrixSize().first; ++j)

{

m\_matrix.CalculateAlgebraicAddition(\*it, j);

}

}

DWORD dwResult = 0;

return dwResult;

}

Matrix.h:

#pragma once

#include "Definitions.h"

class CMatrix

{

public:

CMatrix(const Matrix & inputMatrix);

const Matrix & GetOutputMatrix()const;

void CalculateAlgebraicAddition(size\_t i, size\_t j);

std::pair<int, int> GetInputMatrixSize()const;

double CalcMinor(size\_t iPos, size\_t jPos);

private:

Matrix m\_inputMatrix;

Matrix m\_outputMatrix;

# };

Matrix.cpp

#include "stdafx.h"

#include "Matrix.h"

namespace

{

void GetMatrixWithoutLineAndColumn(const Matrix & mas, Matrix & p, int i, int j, int m) {

int ki, kj, di, dj;

di = 0;

for (ki = 0; ki < m - 1; ki++)

{

if (ki == i)

{

di = 1;

}

dj = 0;

for (kj = 0; kj < m - 1; kj++)

{

if (kj == j) dj = 1;

p[ki][kj] = mas[ki + di][kj + dj];

}

}

}

double CalcDetByGausMethod(Matrix & matrix)

{

size\_t i, j, k;

double det = 1;

double b = 0;

for (i = 0; i < matrix.size(); i++)

{

for (j = i + 1; j < matrix[0].size(); j++)

{

if (matrix[i][i] == 0)

{

if (matrix[i][j] == 0)

{

b = 0;

}

else

{

return 0;

}

}

else

{

b = matrix[j][i] / matrix[i][i];

}

for (k = i; k < matrix.size(); k++)

{

matrix[j][k] = matrix[j][k] - matrix[i][k] \* b;

}

}

det \*= matrix[i][i];

}

return det;

}

}

CMatrix::CMatrix(const Matrix & inputMatrix)

: m\_inputMatrix(inputMatrix)

, m\_outputMatrix(inputMatrix.size(), std::vector<double>(inputMatrix[0].size(), 0))

{

}

const Matrix & CMatrix::GetOutputMatrix() const

{

return m\_outputMatrix;

}

void CMatrix::CalculateAlgebraicAddition(size\_t i, size\_t j)

{

m\_outputMatrix[i][j] = CalcMinor(i, j);

}

std::pair<int, int> CMatrix::GetInputMatrixSize() const

{

return{ m\_inputMatrix.size(), m\_inputMatrix[0].size() };

}

double CMatrix::CalcMinor(size\_t iPos, size\_t jPos)

{

auto matrixSize = m\_inputMatrix.size();

Matrix matrixWithoutIandJ(matrixSize - 1, std::vector<double>(matrixSize - 1));

GetMatrixWithoutLineAndColumn(m\_inputMatrix, matrixWithoutIandJ, iPos, jPos, matrixSize);

auto det = CalcDetByGausMethod(matrixWithoutIandJ);

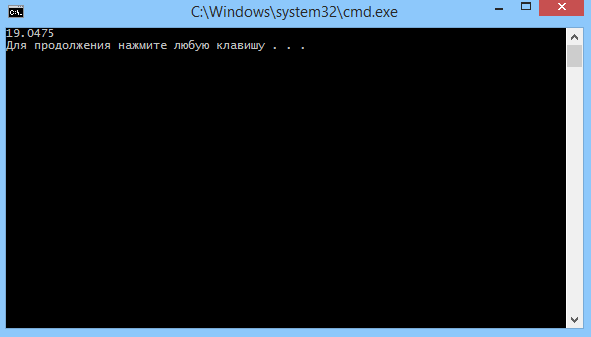
return static\_cast<double>(std::pow(-1, iPos + jPos + 2)) \* (det);

}

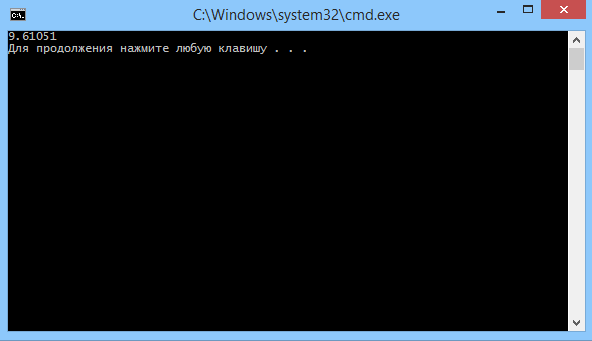
# Вывод

Прежде чем сделать выводы, хотелось бы приложить скриншоты с временем работы программы, выполняющей операции над матрицей, размером 100x100 в режиме Release.

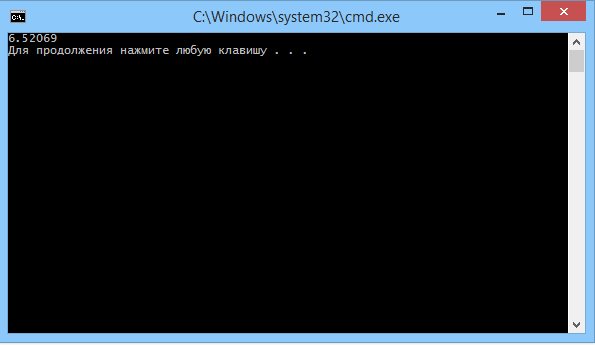
Последовательно:



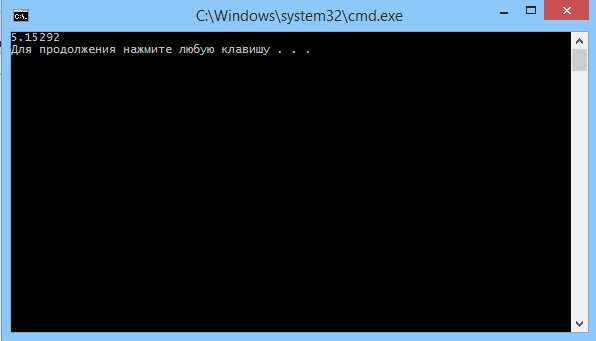
Параллельно (2 потока):



Параллельно (3 потока):



Параллельно (4 потока):



Таким образом, исходя из полученных данных, можно сделать вывод о том, что при параллельном исполнении программы, время выполнения уменьшается.

(после того, как подрефакторю код и запилю графики, сделаю более подробный вывод).