**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»**

Выполнила:

студентка группы ПС-31

факультета Информатики

и Вычислительной Техники

специальности «Программная инженерия»

Моторина Е.Д.

Проверил:

Филимонов А.А.

г. Йошкар-Ола

2016

# Постановка задачи

Определить ранг матрицы.

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

Требования:

* Реализовать последовательный вариант программы для указанного варианта.
* Реализовать параллельный вариант программы. Количество потоков выполнения должно являться входным параметром задачи.
* После завершения программа должна выдавать время своей работы. Подобрать размеры матриц таким образом, чтобы время работы последовательного варианта составляло не менее одной секунды.

# Листинг кода

void WorkWithPart(size\_t startIndex, size\_t endIndex, std::shared\_ptr<Matrix> matrix)

{

auto divisionVector = [&](std::vector<double> &vec, double number)

{

for (auto &it : vec)

{

it /= number;

}

};

auto multiplyVector = [&](std::vector<double> vec, double number)

{

auto returnedVec = vec;

for (auto &it : returnedVec)

{

it \*= number;

}

return returnedVec;

};

auto minusVector = [&](std::vector<double> &vec, std::vector<double> vec1)

{

for (size\_t i = 0; i < vec.size(); ++i)

{

vec[i] -= vec1[i];

}

};

for (size\_t i = startIndex; i < endIndex; ++i)

{

divisionVector((\*matrix)[i], (\*matrix)[i][i]);

for (size\_t j = i + 1; j <matrix->size(); ++j)

{

minusVector((\*matrix)[j], multiplyVector((\*matrix)[i], (\*matrix)[j][i]));

}

}

}

void CParallelWorker::SetThreadCount(size\_t value)

{

m\_threadsCounter = std::max(value, size\_t(1));

m\_threadsCounter = std::min(m\_threadsCounter, size\_t(16));

}

size\_t CParallelWorker::GetRang(Matrix const & matrix)

{

m\_matrix = std::make\_shared<Matrix>(matrix);

m\_threads.clear();

size\_t step = size\_t(m\_matrix->size() / m\_threadsCounter);

m\_threads.resize(m\_threadsCounter);

for (size\_t i = 0; i < m\_threadsCounter; ++i)

{

size\_t startIndex = i \* step;

size\_t endIndex = startIndex + step;

if (i == m\_threadsCounter - 1)

{

endIndex = m\_matrix->size() - 1;

}

m\_threads[i] = std::thread(WorkWithPart, startIndex, endIndex, m\_matrix);

}

for (auto &it : m\_threads)

{

it.join();

}

return CountRang();

}

size\_t CParallelWorker::CountRang()

{

size\_t rang = 0;

for (auto &i : \*m\_matrix)

{

for (auto &j : i)

{

if (j != 0)

{

++rang;

break;

}

}

}

return rang;

}

# Графики

# Вывод

Снимок.JPG

При параллельном методе видно что время работы уменьшилось, примерно в 2 раза (при использовании 16 потоков).