Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

 «Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе №5

на тему: «Решение СЛАУ методом Гаусса с помощью MPI»

Выполнил: студент группы РИС-20-1б

Шумилов Л.С.

Проверил: доцент кафедры ИТАС

Щапов В.А.

Пермь, 2024

**Цель работы** – реализовать алгоритм решения СЛАУ методом Гаусса

**Задание**:

1. Установить MPI;
2. Реализовать алгоритм решения СЛАУ методом Гаусса с использованием библиотеки MPI;

**Ход работы**

Метод Гаусса — это итеративный метод решения систем линейных уравнений. Он заключается в преобразовании исходной системы уравнений к треугольному виду путём элементарных преобразований строк матрицы коэффициентов. Затем решение системы находится путём прямого подстановки.

Краткие шаги метода Гаусса:

1. Перейти к матричному виду системы уравнений.
2. Выбрать главный диагональный элемент матрицы коэффициентов и перевести его в единицу.
3. Вычесть из каждой строки ниже главной строку, умноженную на элемент главной диагонали.
4. Повторять шаги 2 и 3 для каждой следующей главной диагонали.
5. Найти решение системы путём прямой подстановки.

На Листинге 1 представлено решение, аналогично тому, что было реализовано в рамках Лабораторной работы №2, но с использованием MPI.

Листинг 1. Решение СЛАУ с помощью MPI

#include <iostream>

#include <mpi.h>

#define SIZE 2000

#define TOLERANCE 0.000001

double\* new\_random\_matrix() {

double\* matrix = new double[SIZE \* (SIZE + 1)];

for (int i = 0; i < SIZE; i += 1) {

for (int j = 0; j < SIZE; j += 1) {

matrix[i \* SIZE + j] = rand() % 1000 + 1;

}

matrix[i \* SIZE + SIZE] = rand() % SIZE;

}

return matrix;

}

double\* back\_steps(double\* matrix) {

double\* answer = new double[SIZE];

for (int i = SIZE - 1; i >= 0; i -= 1) {

int row\_position = i \* SIZE;

answer[i] = matrix[row\_position + SIZE];

for (int j = i + 1; j < SIZE; j += 1) {

answer[i] -= matrix[row\_position + j] \* answer[j];

}

answer[i] /= matrix[row\_position + i];

}

return answer;

}

void validate\_answer(double\* matrix, double\* answer) {

double sum = 0;

for (int i = 0; i < SIZE; i++)

{

for (int j = 0; j < SIZE; j++)

{

sum += answer[j] \* matrix[i \* SIZE + j];

}

if (std::abs(sum - matrix[i \* SIZE + SIZE]) > TOLERANCE)

std::cerr << "Неверное решение. Разница = " << sum - matrix[i \* SIZE + SIZE] << std::endl;

sum = 0;

}

}

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

double\* matrix = new\_random\_matrix();

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

int rows\_per\_process = SIZE / (size - 1);

double elements = (float)SIZE / (size - 1);

if (rows\_per\_process != elements)

{

std::cerr << "Невозможно вычислить с таким количеством процессов" << std::endl;

return -1;

}

if (rank == 0)

{

double start\_time = MPI\_Wtime();

double\* local\_a = new double[rows\_per\_process \* (SIZE + 1)];

int index = 0;

for (int proc = 1; proc < size; proc++)

{

for (int i = 0; i < rows\_per\_process \* (SIZE + 1); i++)

{

local\_a[i] = matrix[index];

index++;

}

MPI\_Send(local\_a, rows\_per\_process \* (SIZE + 1), MPI\_DOUBLE, proc, 0, MPI\_COMM\_WORLD);

}

index = 0;

for (int proc = 1; proc < size; proc++)

{

MPI\_Recv(local\_a, rows\_per\_process \* (SIZE + 1), MPI\_DOUBLE, proc, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

for (int i = 0; i < rows\_per\_process \* (SIZE + 1); i++)

{

matrix[index] = local\_a[i];

index++;

}

}

double\* answer = back\_steps(matrix);

double end\_time = MPI\_Wtime();

std::cout << "Time = " << end\_time - start\_time << " s" << std::endl;

validate\_answer(matrix, answer);

delete[] local\_a;

}

else

{

int rows\_start = rows\_per\_process \* (rank - 1);

int rows\_end = rank \* rows\_per\_process - 1;

double\* local\_a = new double[rows\_per\_process \* (SIZE + 1)];

double\* local\_b = new double[rows\_per\_process \* (SIZE + 1)];

MPI\_Recv(local\_a, rows\_per\_process \* (SIZE + 1), MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

for (int t = 1; t < rank; t += 1)

{

MPI\_Recv(local\_b, rows\_per\_process \* (SIZE + 1), MPI\_DOUBLE, t, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

for (int i = 0; i < rows\_per\_process; i += 1) {

for (int j = 0; j < rows\_per\_process; j += 1) {

double coefficient = local\_b[i \* SIZE + j + rows\_per\_process \* (t - 1)];

for (int k = SIZE + 1; k >= j; k -= 1) {

local\_a[i \* SIZE + k] -= coefficient \* local\_b[j \* SIZE + k];

}

}

}

}

delete[] local\_b;

int counter = 0;

for (int i = 0; i < rows\_per\_process; i++) {

int row\_position = i \* SIZE;

double coefficient = local\_a[row\_position + i + rows\_start + counter];

for (int j = (SIZE + 1); j >= rows\_start + counter; j -= 1) {

local\_a[row\_position + j] /= coefficient;

}

for (int j = i + 1; j < rows\_per\_process; j += 1) {

coefficient = local\_a[j \* SIZE + rows\_start + counter];

for (int k = SIZE + 1; k >= rows\_start + counter; k -= 1) {

local\_a[j \* SIZE + k] -= coefficient \* local\_a[i \* SIZE + k];

}

}

}

for (int proc = rank + 1; proc < size; proc++)

{

MPI\_Send(local\_a, rows\_per\_process \* (SIZE + 1), MPI\_DOUBLE, proc, 0, MPI\_COMM\_WORLD);

}

MPI\_Send(local\_a, rows\_per\_process \* (SIZE + 1), MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

delete[] local\_a;

}

MPI\_Finalize();

return 0;

}

На Рисунке 1 представлен результат выполнения программы решения СЛАУ 2000 на 2000.



Рисунок . Время на решение СЛАУ методом Гаусса