Министерство науки и высшего образования 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**

по дисциплине **«Технологии блокчейн и распределенные информационные системы»**

**Выполнил** студент гр. РИС-20-2б

Уржумов В.И.

**Проверил** доц. Щапов В.А.

Пермь, 2024 год

**Цели работы**

Рассмотреть способы распараллеливания метода Гаусса для решения систем линейных алгебраических уравнений с помощью MPI.

**Задание к работе**

1. Разработать параллельный алгоритм, написать и отладить параллельную программу решения СЛАУ методом Гаусса в MPI.

**Ход работы**

Алгоритмы решения СЛАУ методом Гаусса в MPI отличаются друг от друга схемой распределения данных по процессам. Матрицу коэффициентов и вектор свободных членом можно распределить широкими горизонтальными полосами. Каждая полоса будет отправлена в соответствующий процесс, то есть первый процесс получит первую полосу, второй вторую, последний последнюю. Такой алгоритм был реализован в данной лабораторной работе.

Напишем на языке программирования C++ программу для решения СЛАУ методом Гаусса с помощью MPI. Функция main представлена на листинге 1:

Листинг 1 – Функция main

#include <mpi.h>

#include <iostream>

#include <cstdlib>

#include <cmath>

const int N = 10;

double RandomDouble(double min, double max) {

return min + (max - min) \* (rand() / static\_cast<double>(RAND\_MAX));

}

double\* FindY(double\* A, double\* X, int n) {

double\* Y = (double\*)malloc(n \* sizeof(double));;

for (int i = 0; i < n; i++) {

Y[i] = 0;

for (int j = 0; j < n; j++) {

Y[i] += A[i \* n + j] \* X[j];

}

}

return Y;

}

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

{

double\* X = new double[N];

int ProcAmout, rank;

MPI\_Status status;

double beginTime, endTime;

MPI\_Init(&argc, &argv);

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

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

int blockSize = N / ProcAmout;

double\* matrixA = NULL;

double\* rightHandB = NULL;

double\* X2 = NULL;

if (rank == 0) {

matrixA = (double\*)malloc(N \* N \* sizeof(double\*));

for (int i = 0; i < N \* N; i++) {

matrixA[i] = RandomDouble(-100, 100);

}

X2 = (double\*)malloc(N \* sizeof(double));

for (int i = 0; i < N; i++) {

X2[i] = RandomDouble(-100, 100);

}

rightHandB = (double\*)malloc(N \* sizeof(double));

rightHandB = FindY(matrixA, X2, N);

}

double\* subA = (double\*)malloc(blockSize \* N \* sizeof(double\*));

double\* subB = (double\*)malloc(blockSize \* sizeof(double));

MPI\_Scatter(matrixA, blockSize \* N, MPI\_DOUBLE, subA, blockSize \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Scatter(rightHandB, blockSize, MPI\_DOUBLE, subB, blockSize, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double\* row = (double\*)malloc(N \* sizeof(double));

double b;

int column;

if (rank == 0) {

beginTime = MPI\_Wtime();

for (int k = 0; k < blockSize; k++)

{

column = rank \* blockSize + k;

for (int i = column + 1; i < N; i++) {

subA[k \* N + i] = subA[k \* N + i] / subA[k \* N + column];

}

subB[k] /= subA[k \* N + column];

subA[k \* N + column] = 1;

b = subB[k];

memcpy(row, &subA[k \* N], sizeof(double) \* N);

MPI\_Bcast(row, N, MPI\_DOUBLE, rank, MPI\_COMM\_WORLD);

MPI\_Bcast(&b, 1, MPI\_DOUBLE, rank, MPI\_COMM\_WORLD);

for (int j = k + 1; j < blockSize; j++) {

for (int i = column + 1; i < N; i++) {

subA[j \* N + i] = subA[j \* N + i] - subA[j \* N + k] \* row[i];

}

subB[j] -= subA[j \* N + k] \* b;

subA[j \* N + column] = 0;

}

}

}

else {

for (int k = 0; k < rank \* blockSize; k++) {

MPI\_Bcast(row, N, MPI\_DOUBLE, k / blockSize, MPI\_COMM\_WORLD);

MPI\_Bcast(&b, 1, MPI\_DOUBLE, k / blockSize, MPI\_COMM\_WORLD);

for (int j = 0; j < blockSize; j++) {

for (int i = k + 1; i < N; i++) {

subA[j \* N + i] = subA[j \* N + i] - subA[j \* N + k] \* row[i];

}

subB[j] -= subA[j \* N + k] \* b;

subA[j \* N + k] = 0;

}

}

for (int k = 0; k < blockSize; k++)

{

column = rank \* blockSize + k;

for (int i = column + 1; i < N; i++) {

subA[k \* N + i] = subA[k \* N + i] / subA[k \* N + column];

}

subB[k] /= subA[k \* N + column];

subA[k \* N + column] = 1;

b = subB[k];

memcpy(row, &subA[k \* N], sizeof(double) \* N);

MPI\_Bcast(row, N, MPI\_DOUBLE, rank, MPI\_COMM\_WORLD);

MPI\_Bcast(&b, 1, MPI\_DOUBLE, rank, MPI\_COMM\_WORLD);

for (int j = k + 1; j < blockSize; j++) {

for (int i = column + 1; i < N; i++) {

subA[j \* N + i] = subA[j \* N + i] - subA[j \* N + column] \* row[i];

}

subB[j] -= subA[j \* N + column] \* b;

subA[j \* N + column] = 0;

}

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Gather(subA, blockSize \* N, MPI\_DOUBLE, matrixA, blockSize \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Gather(subB, blockSize, MPI\_DOUBLE, rightHandB, blockSize, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

for (int k = N - 1; k >= 0; k--) {

X[k] = rightHandB[k];

for (int i = N - 1; i > k; i--) {

X[k] = X[k] - matrixA[k \* N + i] \* X[i];

}

X[k] = X[k] / matrixA[k \* N + k];

}

endTime = MPI\_Wtime();

std::cout << "Time = " << endTime - beginTime << std::endl;

}

MPI\_Finalize();

if (rank == 0) {

int c = 0;

for (int i = 0; i < N; i++) {

if (std::abs(X[i] - X2[i]) > 0.00001) {

c++;

}

}

std::cout << "Number of non - matching X = " << c << std::endl;

}

}

Процессор с рангом 0 принимает на себя роль мастера, который распределяет данные между процессами с помощью функции MPI\_Scatter. Предполагается, что количество строк матрицы свободных членов делится без остатка на количество процессов.

Для автоматизации проверки правильности решения СЛАУ методом Гаусса, матрица весов и вектор неизвестных генерируются автоматически, после чего находится соответствующий вектор свободных членов.

После завершения обратного хода метода Гаусса, полученный вектор неизвестных сравнивается с генерированным в самом начале вектором неизвестных. Если количество не совпадающих значений равно 0, то метод сработал правильно.

Результат работы программы представлен на рисунке 1.



Рисунок 1 – Выполнение программы

На рисунке 1 представлен запуск программы с матрицей размерности 4800 на 4800 и с различным числом процессов. Лучший показатель времени достигается при использовании 8 процессов, что соответствует числу вычислительных ядер использованного процессора.

**Заключение**

В ходе выполнения лабораторной работы была написана программа для решения системы линейных алгебраических уравнений с помощью метода Гаусса. Применена библиотека MPI для распараллеливания метода Гаусса. Цели работы достигнуты, все задачи выполнены.