Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа №2**

**«Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI»**

**по курсу: «Разработка параллельных и распределенных программ»**

Выполнил:

Студент группы ИУ9-52Б

Марченко А.И.

Проверил:

Царев А.С.

**1. Цель работы**

Решить систему уравнений одним из методов.

Мой вариант - 2

**2. Реализация**

Код программы:

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <cassert>  
#include "mpich/mpi.h"  
#include "mpich/mpi\_proto.h"  
#include <chrono>  
  
double norm(std::vector <double> v){  
 double sum = 0;  
 for (auto elem : v){  
 sum += elem \* elem;  
 }  
 return sqrt(sum);  
}  
  
double vector\_mult(std::vector <double> a, std::vector <double> b){  
 assert(a.size() == b.size());  
 double res = 0;  
 for (int i = 0; i < a.size(); i++){  
 res += a[i] \* b[i];  
 }  
 return res;  
}  
  
void mult(std::vector <std::vector <double> > A, std::vector <double> x, int raw1, int raw2, std::vector <double>& res){  
 double sum;  
 for (int i = raw1; i <= raw2; i++){  
 sum = 0;  
 for (int j = 0; j < x.size(); j++){  
 sum += A[i][j] \* x[j];  
 }  
 res[i] = sum;  
 }  
}  
std::vector <double> mult\_digit\_vector(double a, std::vector <double> b) {  
 for (int i = 0; i < b.size(); i++){  
 b[i] \*= a;  
 }  
 return b;  
}  
  
std::vector <double> vector\_diff(std::vector <double> a, std::vector <double> b) {  
 assert(a.size() == b.size());  
 for (int i = 0; i < a.size(); i++){  
 a[i] -= b[i];  
 }  
 return a;  
}  
  
  
int main(int argc, char \*\*argv) {  
 int matrix\_size = 500;  
 //std::cin >> matrix\_size;  
  
 std::vector <std::vector <double> > A(matrix\_size, std::vector <double> (matrix\_size));  
 for (int i = 0; i < A.size(); i++){  
 for (int j = 0; j < A[0].size(); j++){  
 if (i == j) A[i][j] = 2.0;  
 else A[i][j] = 1.0;  
 }  
 }  
 std::vector <double> x (matrix\_size, 5);  
 int rank = -1, col\_ranks = -1;  
  
 uint n = A.size(), m = A[0].size();  
 //std::vector <double> b(m, n+1);  
 std::vector <double> res(n, 0.0);  
 std::vector <double> res2(n, 0.0);  
 std::vector <double> Ay(n, 0.0);  
 std::vector <double> b (n);  
 for (double & i : b){  
 i = n+1;  
 }  
  
 int chunk\_size, bonus;  
 std::vector <std::pair <int, int> > intervals;  
 double eps = 0.00001;  
  
 MPI\_Init(&argc, &argv);  
  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &col\_ranks);  
  
 auto go = std::chrono::high\_resolution\_clock::now();  
  
 chunk\_size = A.size() / col\_ranks, bonus = A.size() - chunk\_size \* col\_ranks;  
 for (int start = 0, end = chunk\_size; start < A.size(); start = end, end = start + chunk\_size) {  
 if (bonus) {  
 end++;  
 bonus--;  
 }  
 intervals.emplace\_back(start, end-1);  
 }  
 std::vector <double> send\_res (n);  
 std::vector <double> collect\_res (n);  
 std::vector <double> y (n);  
 for (int k = 0; k < 30; k++) {  
 if (rank == 0) {  
 double tau, error;  
  
 mult(A, x, intervals[rank].first, intervals[rank].second, res);  
 send\_res = x;  
 for (int i = 1; i < col\_ranks; i++) {  
 MPI\_Send(&send\_res[0], send\_res.size(), MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);  
 }  
  
 for (int i = 1; i < col\_ranks; i++) {  
 MPI\_Recv(&collect\_res[0], n, MPI\_DOUBLE, i, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  
 for (int j = intervals[i].first; j <= intervals[i].second; j++) {  
 res[j] = collect\_res[j];  
 }  
 }  
 y = vector\_diff(res, b);  
  
 mult(A, y, intervals[rank].first, intervals[rank].second, Ay);  
 send\_res = x;  
  
 for (int i = 1; i < col\_ranks; i++) {  
 MPI\_Send(&send\_res[0], send\_res.size(), MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);  
 }  
 for (int i = 1; i < col\_ranks; i++) {  
 MPI\_Recv(&collect\_res[0], n, MPI\_DOUBLE, i, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  
 for (int j = intervals[i].first; j <= intervals[i].second; j++) {  
 Ay[j] = collect\_res[j];  
 }  
 }  
  
 tau = vector\_mult(y, Ay) / vector\_mult(Ay, Ay);  
  
 x = vector\_diff(x, mult\_digit\_vector(tau, y));  
  
 mult(A, x, intervals[rank].first, intervals[rank].second, res2);  
 send\_res = x;  
 for (int i = 1; i < col\_ranks; i++) {  
 MPI\_Send(&send\_res[0], send\_res.size(), MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);  
 }  
 for (int i = 1; i < col\_ranks; i++) {  
 MPI\_Recv(&collect\_res[0], n, MPI\_DOUBLE, i, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  
  
 for (int j = intervals[i].first; j <= intervals[i].second; j++) {  
 res2[j] = collect\_res[j];  
 }  
 }  
  
// for (auto i : x){  
// std::cout << i << " ";  
// }  
// std::cout << std::endl;  
  
 error = norm(vector\_diff(res2, b)) / norm(b);  
  
 if (error < eps) {  
 std::cout << "Result: " << std::endl;  
// for (double re : x){ ВЫВОД УБРАН, ТАК КАК ОН ЛОМАЕТ ПРОЦЕССЫ НА БОЛЬШИХ МАТРИЦАХ  
// std::cout << re << " ";  
// }  
 std::cout << std::endl;  
 fflush(stdout);  
  
 MPI\_Finalize();  
 break;  
 }  
 }  
 }  
 for (int k = 0; k < 30; k++) {  
 if (rank != 0) {  
 MPI\_Recv(&send\_res[0], n, MPI\_DOUBLE, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  
 mult(A, send\_res, intervals[rank].first, intervals[rank].second, collect\_res);  
 MPI\_Send(&collect\_res[0], n, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);  
 }  
 }  
 auto finish = std::chrono::high\_resolution\_clock::now();  
  
 auto duration = duration\_cast<std::chrono::microseconds>(finish - go);  
 std::cout << "Time: "<< duration.count() << " ms" << std::endl;  
  
 std::cout << "Result: " << std::endl;  
 if (rank == 0)  
 for (double re : x){  
 std::cout << re << " ";  
 }  
}  
/\* 1: 9.2  
 \* 2: 8.2  
 \* 3: 7.8  
 \* 4: 7.6  
 \* 5: 7.5  
 \*/

**3. Характеристики устройства:**

AMD Ryzen 5 4600HS

6 ядер, 12 потоков

**4. Время работы:**

/\* 1: 9.2  
 \* 2: 8.2  
 \* 3: 7.8  
 \* 4: 7.6  
 \* 5: 7.5

График: