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 №3**

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

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

Выполнил:

Студент группы ИУ9-52Б

Марченко А.И.

Проверил:

Царев А.С.

**1. Цель работы**

Переписать задание лаб. работы №2 на технологии Open MPI.

**2. Реализация**

Код программы:

#include <omp.h>  
#include <iostream>  
#include <vector>  
#include <cmath>  
#include <cassert>  
#include <random>  
#include <chrono>  
#include <ratio>  
  
double norm(const 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){  
 int i, j;  
 omp\_set\_num\_threads(5);  
 #pragma omp parallel for shared(A, x, res, raw1, raw2) private(i, j)  
 for (i = raw1; i <= raw2; i++){  
 res[i] = 0;  
 for (j = 0; j < x.size(); j++){  
 res[i] += A[i][j] \* x[j];  
 }  
 }  
}  
  
void default\_mult(std::vector <std::vector <double> > A, std::vector <double> x, int raw1, int raw2, std::vector <double>& res){  
 double sum;  
 int i, j;  
 for (i = raw1; i <= raw2; i++){  
 sum = 0;  
 for (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) {  
 int i = 0;  
 #pragma omp parallel for shared(a, b) private(i)  
 for (int i = 0; i < b.size(); i++){  
 b[i] \*= a;  
 }  
 return b;  
}  
  
std::vector <double> mult\_digit\_vector\_default(double a, std::vector <double> b) {  
 int i = 0;  
 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;  
}  
  
void printMatrix(const std::vector <std::vector <double> >& mt){  
 for (int i = 0; i < mt.size(); i++){  
 for (int j = 0; j < mt[0].size(); j++){  
 std::cout << mt[i][j] << " ";  
 }  
 std::cout << std::endl;  
 }  
}  
  
void fillMatrixWithRandom(std::vector <std::vector <double> >& mt) {  
 std::random\_device rd;  
 std::mt19937 gen(rd());  
 for (int i = 0; i < mt.size(); i++){  
 for (int j = 0; j < mt[0].size(); j++){  
 mt[i][j] = (1.0 + gen() % 100) / 10.f;  
 }  
 }  
}  
  
std::vector <double> solve(std::vector <std::vector <double> > A, double eps){  
 int n = A.size(), m = A[0].size();  
 std::vector <double> x(m, 10);  
 //std::vector <double> b(m, n+1);  
 std::vector <double> b (n);  
 for (double & i : b){  
 i = n+1;  
 }  
 std::vector <double> res(m, 0);  
 std::vector <double> Ay(m, 0);  
  
 double tau, error;  
 int iter = 1;  
 while(true){  
 mult(A, x, 0, n-1, res);  
  
 auto y = vector\_diff(res, b);  
 mult(A, y, 0, n-1, Ay);  
 tau = vector\_mult(y, Ay) / vector\_mult(Ay, Ay);  
 x = vector\_diff(x, mult\_digit\_vector(tau, y));  
  
 std::cout << "tau: " << tau << std::endl;  
  
 mult(A, x, 0, n-1, res);  
  
 assert(norm(b) != 0);  
  
 error = norm(vector\_diff(res, b)) / norm(b);  
 std::cout << iter << " " << error << std::endl;  
 iter++;  
  
 if (error < eps) break;  
 }  
  
 return x;  
}  
  
std::vector <double> solve\_without\_parallel(std::vector <std::vector <double> > A, double eps){  
 int n = A.size(), m = A[0].size();  
 std::vector <double> x(m, 8.1);  
 //std::vector <double> b(m, n+1);  
 std::vector <double> b (n);  
 for (double & i : b){  
 i = n+1;  
 }  
 std::vector <double> res(m, 0);  
 std::vector <double> Ay(m, 0);  
  
 double tau, error;  
 int iter = 1;  
 while(true){  
 default\_mult(A, x, 0, n-1, res);  
  
 auto y = vector\_diff(res, b);  
 default\_mult(A, y, 0, n-1, Ay);  
 tau = vector\_mult(y, Ay) / vector\_mult(Ay, Ay);  
 x = vector\_diff(x, mult\_digit\_vector\_default(tau, y));  
  
 std::cout << "tau: " << tau << std::endl;  
  
 default\_mult(A, x, 0, n-1, res);  
  
 assert(norm(b) != 0);  
  
 error = norm(vector\_diff(res, b)) / norm(b);  
 std::cout << iter << " " << error << std::endl;  
 iter++;  
  
 if (error < eps) break;  
 }  
  
 return x;  
}  
  
int main(int argc, char\*\* argv) {  
 int matrix\_sixe = 10000;  
 std::vector <std::vector <double> > a(matrix\_sixe, std::vector <double> (matrix\_sixe));  
 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;  
 }  
 }  
  
 auto go = std::chrono::high\_resolution\_clock::now();  
 auto res = solve(a, 0.000001);  
 auto finish = std::chrono::high\_resolution\_clock::now();  
  
 auto duration = std::chrono::duration\_cast <std::chrono::microseconds> (finish - go);  
 std::cout << duration.count() << " ms" << std::endl;  
  
// auto go = std::chrono::high\_resolution\_clock::now();  
// auto res1 = solve\_without\_parallel(a, 0.000001);  
// auto finish = std::chrono::high\_resolution\_clock::now();  
//  
// auto duration = std::chrono::duration\_cast <std::chrono::microseconds> (finish - go);  
 std::cout << duration.count() << " ms" << std::endl;  
  
}

**3. Характеристики устройства:**

AMD Ryzen 5 4600HS

6 ядер, 12 потоков

**4. Время работы:**

(Измерения на матрице 10000х10000)

1 поток: 5.1 сек

2 потока: 3.3 сек

3 потока: 3.4 сек (видимо совсем неудачное число потоков для этих данных)

4 потока: 2.6 сек

5 потоков: 2.3 сек

6 потоков: 2.1 сек

7 потоков: 2.4 (дальше нет улучшений)

График: