**Компиляция и запуск:**

**mpicxx file**

**mpiexec [–n threads] ./file.out**

**Пример кода с разрезанием матрицы А:**

#include <stdio.h>

#include <iostream>

#include <time.h>

#include <math.h>

#include <mpi.h>

#define N 10000

double norma(double arr[], int size) {

double result = 0;

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

result += arr[i] \* arr[i];

return result;

}

double A[N \* N];

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

MPI\_Init(&argc, &argv);

int start, end;

start = time(NULL);

int rank, size;

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

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

int\* scounts\_str = NULL;

int\* displs\_str = NULL;

int\* scounts = NULL;

int\* displs = NULL;

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

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

A[i \* N + j] = 1;

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

A[i \* N] = 2;

scounts\_str = new int[size];

scounts = new int[size];

displs\_str = new int[size];

displs = new int[size];

int cnt = N / size;

int rmd = N % size;

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

scounts\_str[i] = cnt \* N;

scounts[i] = cnt;

if (rmd > 0) {

scounts\_str[i] += N;

scounts[i] += 1;

rmd--;

}

}

displs\_str[0] = 0;

displs[0] = 0;

for (int i = 1; i < size; i++) {

displs\_str[i] = displs\_str[i - 1] + scounts\_str[i - 1];

displs[i] = displs[i - 1] + scounts[i - 1];

}

double\* rvbuf = new double[scounts\_str[rank]];

MPI\_Scatterv(A, scounts\_str, displs\_str, MPI\_DOUBLE, rvbuf, scounts\_str[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double b[N];

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

b[i] = N + 1;

double x[N] = { 0 };

double local\_estimate;

double estimate;

double normab = norma(b, N);

int usize = scounts[rank];

double vec\_u[N];

double\* u = new double[usize];

while (1) {

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

u[i] = 0;

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

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

u[i] += rvbuf[i \* N + j] \* x[j];

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

u[i] -= b[i];

local\_estimate = norma(u, usize) / normab;

MPI\_Allreduce(&local\_estimate, &estimate, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);

if (estimate < 0.0000000001)

break;

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

u[i] \*= 0.00001;

MPI\_Allgatherv(u, scounts[rank], MPI\_DOUBLE, vec\_u, scounts, displs, MPI\_DOUBLE, MPI\_COMM\_WORLD);

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

x[i] -= vec\_u[i];

}

end = time(NULL);

if (rank == 0)

std::cout << end - start << std::endl;

MPI\_Finalize();

delete[] scounts\_str;

delete[] displs\_str;

delete[] scounts;

delete[] displs;

delete[] rvbuf;

delete[] u;

return 0;

}

**Пример кода с разрезанием матрицы А и векторов x, b:**

#include <stdio.h>

#include <iostream>

#include <time.h>

#include <math.h>

#include <mpi.h>

#define N 10000

double norma(double arr[], int size) {

double result = 0;

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

result += arr[i] \* arr[i];

return result;

}

double A[N \* N];

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

MPI\_Init(&argc, &argv);

int start, end;

start = time(NULL);

int rank, size;

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

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

int\* scounts\_str = NULL;

int\* displs\_str = NULL;

int\* scounts = NULL;

int\* displs = NULL;

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

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

A[i \* N + j] = 1;

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

A[i \* N] = 2;

scounts\_str = new int[size];

scounts = new int[size];

displs\_str = new int[size];

displs = new int[size];

int cnt = N / size;

int rmd = N % size;

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

scounts\_str[i] = cnt \* N;

scounts[i] = cnt;

if (rmd > 0) {

scounts\_str[i] += N;

scounts[i] += 1;

rmd--;

}

}

displs\_str[0] = 0;

displs[0] = 0;

for (int i = 1; i < size; i++) {

displs\_str[i] = displs\_str[i - 1] + scounts\_str[i - 1];

displs[i] = displs[i - 1] + scounts[i - 1];

}

double\* rvbuf = new double[scounts\_str[rank]];

MPI\_Scatterv(A, scounts\_str, displs\_str, MPI\_DOUBLE, rvbuf, scounts\_str[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double b[N];

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

b[i] = N + 1;

double\* drob\_b = new double[scounts[rank]];

MPI\_Scatterv(b, scounts, displs, MPI\_DOUBLE, drob\_b, scounts[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double\* drobx = new double[scounts[rank]];

for (int i = 0; i < scounts[rank]; i++)

drobx[i] = 0;

double local\_estimate;

double estimate;

double normab = norma(b, N);

int usize = scounts[rank];

double\* u = new double[usize];

double\* tmp\_string = new double[N];

double\* buffer = new double[scounts[0]];

while (1) {

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

u[i] = 0;

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

if (rank == i) {

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

buffer[j] = drobx[j];

}

MPI\_Bcast(buffer, scounts[0], MPI\_DOUBLE, i, MPI\_COMM\_WORLD);

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

for (int k = 0; k < scounts[i]; k++) {

u[j] += rvbuf[j \* N + displs[i] + k] \* buffer[k];

}

}

}

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

u[i] -= drob\_b[i];

local\_estimate = norma(u, usize) / normab;

MPI\_Allreduce(&local\_estimate, &estimate, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);

if (estimate < 0.0000000001)

break;

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

u[i] \*= 0.00001;

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

drobx[i] -= u[i];

}

MPI\_Allgatherv(drobx, scounts[rank], MPI\_DOUBLE, tmp\_string, scounts, displs, MPI\_DOUBLE, MPI\_COMM\_WORLD);

end = time(NULL);

if (rank == 0) {

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

printf("%lf ", tmp\_string[i]);

}

std::cout << end - start << std::endl;

}

MPI\_Finalize();

delete[] drob\_b;

delete[] scounts\_str;

delete[] displs\_str;

delete[] scounts;

delete[] displs;

delete[] rvbuf;

delete[] u;

delete[] drobx;

delete[] tmp\_string;

delete[] buffer;

return 0;

}