Листинг

```
#include <mpi.h>
#include <iostream>
#include <fstream>


int main(int argc, char* argv[]) {
    int procs_count, procs_rank;
    const int n = 10;
    MPI_Status status;
    double t1, t2, t3, t4, t;
    double *m = new double[n * n];
    std::ofstream fout("input.txt");
    std::ofstream out("output.txt");
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &procs_count);
    MPI_Comm_rank(MPI_COMM_WORLD, &procs_rank);
    if (procs_rank == 0) {
        srand(0);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                m[i * n + j] = rand();
            }
            fout << "\n";
        }
        fout.close();
        std::cout << "Decomposition started.\n";
    }
    t1 = MPI_Wtime();
    MPI_Bcast(m, n * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    t2 = MPI_Wtime();
    for (int k = 0; k < n - 1; k++) {
        if (k % procs_count == procs_rank) {
            for (int i = k + 1; i < n; i++) {
                m[k * n + i] /= m[k * n + k];
            }
        }
        MPI_Bcast(&(m[k * n + k + 1]), n - k - 1, MPI_DOUBLE,
                  k % procs_count, MPI_COMM_WORLD);
        for (int i = k + 1; i < n; i++) {
            if (i % procs_count == procs_rank) {
                for (int j = k + 1; j < n; j++) {
                    m[i * n + j] -= m[i * n + k] * m[k * n + j];
                }
            }
        }
    }
    t3 = MPI_Wtime();
    if (procs_rank == 0) {
        out << "Matrix L:\n";
        for (int i = 0; i < n; i++) {
            if (i % procs_count != 0) {
                MPI_Recv(&(m[i * n]), n, MPI_DOUBLE,
                         i % procs_count, 100, MPI_COMM_WORLD, &status);
            }
            for (int j = 0; j <= i; j++) {
                out << m[i * n + j] << " ";
```

```
            }
            for (int j = i + 1; j < n; j++) {
                out << "0 ";
            }
            out << "\n";
        }
        out << "\nMatrix U:\n";
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                out << "0 ";
            }
            out << "1 ";
            for (int j = i + 1; j < n; j++) {
                out << m[i * n + j] << " ";
            }
            out << "\n";
        }
        out.close();
    } else {
        for (int i = procs_rank; i < n; i += procs_count) {
            MPI_Send(&(m[i * n]), n, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD);
        }
    }
    t = t3 - t2;
    if (procs_rank == 0) {
        std::cout << "Time to share matrix: " << (t2 - t1) * 1000 << " ms.\n";
        std::cout << "Time for calculations, proc 0: " << t * 1000 << " ms.\n";
        for (int i = 1; i < procs_count; i++) {
            MPI_Recv(&t, 1, MPI_DOUBLE, i, 101, MPI_COMM_WORLD, &status);
            std::cout << "Time for calculations, proc "
                      << i << ": " << t * 1000 << " ms.\n";
        }
        std::cout << "Program worked for : " << (t3 - t1) * 1000 << " ms.\n";
    } else {
        MPI_Send(&t, 1, MPI_DOUBLE, 0, 101, MPI_COMM_WORLD);
    }
    delete[] m;
    MPI_Finalize();
    return 0;
}
```

Время выполнения

Размер матрицы	Число ядер	Время выполнения(мс)
10	1	0.0040207
	2	0.0285756
	4	0.0895746
	8	0.137309
100	1	1.60711
	2	0.88601
	4	1.2991
	8	1.31596
1000	1	1429.75
	2	573.481
	4	375.678
	8	679.075