

Лабораторная Работа № 1

Умножение матрицы на вектор

Царик Виталий
3-й курс 2-я группа

| n | Процессов | Время выполнения, с |
|------|-----------|---------------------|
| 10 | 1 | 0.0040 |
| 10 | 5 | 0.0060 |
| 10 | 10 | 0.0055 |
| 100 | 1 | 0.0678 |
| 100 | 5 | 0.0605 |
| 100 | 10 | 0.041769 |
| 1000 | 1 | 0.002617 |
| 1000 | 5 | 0.0051708 |
| 1000 | 10 | 0.007129 |

Таблица 1: Результаты для размерности n

Листинг 1: файл lab1.cpp

```
1 #include <fstream>
2 #include <iostream>
3
4 #include <mpi.h>
5
6 #include "MPI.h"
7
8 using namespace std;
9
10 int* calc(int** rows, int rows_number, int n, int* b)
11 {
12     int* result = new int[rows_number];
13
14     for (int i = 0; i < rows_number; ++i)
15     {
16         result[i] = 0;
17         for (int j = 0; j < n; ++j)
18             result[i] += rows[i][j] * b[j];
19     }
20     return result;
```

```

21 }
22
23 int main()
24 {
25     auto process = MPI();
26
27     int* b = nullptr;
28     int** a = nullptr;
29     int* send_rows = nullptr;
30     int* steps = nullptr;
31     int n = 0;
32     const int size = process.get_size();
33     const int rank = process.get_rank();
34
35     int process_rows_number = 0;
36     int rest_size = 0;
37
38     int* rel_beg = nullptr;
39
40     double start_time, end_time;
41
42
43     if (rank == MPI::MASTER_RANK)
44     {
45         start_time = process.time();
46         int** A = nullptr;
47         ifstream fin;
48         fin.open("input.txt");
49
50         fin >> n;
51
52         process.send_to_others(&n, 1, MPI_INT);
53
54         A = new int* [n];
55         for (int i = 0; i < n; ++i)
56             A[i] = new int[n];
57
58         b = new int[n];
59
60         for (int i = 0; i < n; ++i)
61             for (int j = 0; j < n; j++)
62                 fin >> A[i][j];
63         for (int i = 0; i < n; ++i)
64             fin >> b[i];
65
66         int rows_number = n / size;
67         rest_size = n % size;
68
69         send_rows = new int[size];
70         for (int i = 0; i < size; ++i)
71         {
72             send_rows[i] = rows_number;
73             if (rest_size)
74                 {

```

```

75     send_rows[i]++;
76     rest_size--;
77 }
78 }
79
80
81
82 rel_beg = new int[size];
83 for (int i = 0; i < size; ++i)
84 {
85     rel_beg[i] = i ? rel_beg[i - 1] + send_rows[i - 1] : 0;
86     if (i != MPI::MASTER_RANK)
87     {
88         process.send(&send_rows[i], 1, MPI_INT, i);
89         for (int j = 0; j < send_rows[i]; ++j)
90             process.send(A[rel_beg[i] + j], n, MPI_INT, i);
91     }
92     else
93     {
94         process_rows_number = send_rows[i];
95         a = new int* [process_rows_number];
96         for (int j = 0; j < process_rows_number; ++j)
97             a[j] = A[rel_beg[i] + j];
98     }
99 }
100 process.send_to_others(b, n, MPI_INT);
101 }
102 else {
103     process.receive(&n, 1, MPI_INT, MPI::MASTER_RANK);
104
105     process.receive(&process_rows_number, 1, MPI_INT, MPI::MASTER_RANK);
106     a = new int* [process_rows_number];
107     for (int i = 0; i < process_rows_number; ++i)
108     {
109         a[i] = new int[n];
110         process.receive(a[i], n, MPI_INT, MPI::MASTER_RANK);
111     }
112
113     b = new int[n];
114     process.receive(b, n, MPI_INT, MPI::MASTER_RANK);
115 }
116
117 int* local_result = calc(a, process_rows_number, n, b);
118
119 if (rank != MPI::MASTER_RANK)
120 {
121     process.send(local_result, process_rows_number, MPI_INT, MPI::
122         MASTER_RANK);
123 }
124 else
125 {
126     int* result = new int[n];
127     for (int i = 0; i < size; ++i)

```

```

128     if (i != MPI::MASTER_RANK)
129         process.receive(result + rel_beg[i], send_rows[i], MPI_INT, i);
130     else
131     {
132         for (int j = 0; j < process_rows_number; ++j)
133             result[rel_beg[i] + j] = local_result[j];
134     }
135
136     ofstream fout("output.txt");
137     cout << "Result vector: ";
138     for (int i = 0; i < n; ++i)
139     {
140         fout << result[i] << ' ';
141         cout << result[i] << ' ';
142     }
143
144     fout.close();
145     end_time = process.time();
146     cout << "\nTime: " << end_time - start_time;
147 }
148
149 return 0;
150 }

```

Листинг 2: файл MPI.h

```

1 #pragma once
2 #include <mpi.h>
3
4 class MPI
5 {
6 public:
7     const static size_t MASTER_RANK = 0;
8
9     MPI()
10    {
11        MPI_Init(nullptr, nullptr);
12        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
13        MPI_Comm_size(MPI_COMM_WORLD, &size);
14    }
15    ~MPI()
16    {
17        MPI_Finalize();
18    }
19
20    void send(void* buffer, int count, MPI_Datatype datatype, int dest)
21    {
22        MPI_Send(buffer, count, datatype, dest, 98, MPI_COMM_WORLD);
23    }
24
25    void send_to_others(void* buffer, int count, MPI_Datatype datatype)
26    {
27        for (int i = 0; i < size; ++i)
28            if (i != rank)

```

```

29         send(buffer, count, datatype, i);
30     }
31
32     void receive(void* buffer, int count, MPI_Datatype datatype, int source)
33     {
34         MPI_Recv(buffer, count, datatype, source, 98, MPI_COMM_WORLD, new
35             MPI_Status);
36     }
37
38     static auto time()
39     {
40         return MPI_Wtime();
41     }
42
43     int get_size() const
44     {
45         return size;
46     }
47
48     int get_rank() const
49     {
50         return rank;
51     }
52 private:
53     int size;
54     int rank;
55 };

```