# Лабараторная 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

```
#include <fstream>
#include <iostream>

#include <mpi.h>

#include "MPI.h"

using namespace std;

int* calc(int** rows, int rows_number, int n, int* b)
{
  int* result = new int[rows_number];

  for (int i = 0; i < rows_number; ++i)
  {
    result[i] = 0;
    for (int j = 0; j < n; ++j)
      result[i] += rows[i][j] * b[j];
  }
  return result;
```

```cpp
}

int main()
{
  auto process = MPI();

  int* b = nullptr;
  int** a = nullptr;
  int* send_rows = nullptr;
  int* steps = nullptr;
  int n = 0;
  const int size = process.get_size();
  const int rank = process.get_rank();

  int process_rows_number = 0;
  int rest_size = 0;

  int* rel_beg = nullptr;

  double start_time, end_time;


  if (rank == MPI::MASTER_RANK)
  {
    start_time = process.time();
    int** A = nullptr;
    ifstream fin;
    fin.open("input.txt");

    fin >> n;

    process.send_to_others(&n, 1, MPI_INT);

    A = new int* [n];
    for (int i = 0; i < n; ++i)
      A[i] = new int[n];

    b = new int[n];

    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; j++)
        fin >> A[i][j];
    for (int i = 0; i < n; ++i)
      fin >> b[i];

    int rows_number = n / size;
    rest_size = n % size;

    send_rows = new int[size];
    for (int i = 0; i < size; ++i)
    {
      send_rows[i] = rows_number;
      if (rest_size)
      {
        send_rows[i]++;
        rest_size--;
      }
    }



    rel_beg = new int[size];
    for (int i = 0; i < size; ++i)
    {
      rel_beg[i] = i ? rel_beg[i - 1] + send_rows[i - 1] : 0;
      if (i != MPI::MASTER_RANK)
      {
        process.send(&send_rows[i], 1, MPI_INT, i);
        for (int j = 0; j < send_rows[i]; ++j)
          process.send(A[rel_beg[i] + j], n, MPI_INT, i);
      }
      else
      {
        process_rows_number = send_rows[i];
        a = new int* [process_rows_number];
        for (int j = 0; j < process_rows_number; ++j)
          a[j] = A[rel_beg[i] + j];
      }
    }
    process.send_to_others(b, n, MPI_INT);
  }
  else {
    process.receive(&n, 1, MPI_INT, MPI::MASTER_RANK);

    process.receive(&process_rows_number, 1, MPI_INT, MPI::MASTER_RANK);
    a = new int* [process_rows_number];
    for (int i = 0; i < process_rows_number; ++i)
    {
      a[i] = new int[n];
      process.receive(a[i], n, MPI_INT, MPI::MASTER_RANK);
    }

    b = new int[n];
    process.receive(b, n, MPI_INT, MPI::MASTER_RANK);
  }

  int* local_result = calc(a, process_rows_number, n, b);

  if (rank != MPI::MASTER_RANK)
  {
    process.send(local_result, process_rows_number, MPI_INT, MPI::
       MASTER_RANK);
  }
  else
  {
    int* result = new int[n];

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

```cpp
      if (i != MPI::MASTER_RANK)
        process.receive(result + rel_beg[i], send_rows[i], MPI_INT, i);
      else
      {
        for (int j = 0; j < process_rows_number; ++j)
          result[rel_beg[i] + j] = local_result[j];
      }

    ofstream fout("output.txt");
    cout << "Result vector: ";
    for (int i = 0; i < n; ++i)
    {
      fout << result[i] << ' ';
      cout << result[i] << ' ';
    }

    fout.close();
    end_time = process.time();
    cout << "\nTime: " << end_time - start_time;
  }

  return 0;
}
```

Листинг 2: файл MPI.h

```cpp
#pragma once
#include <mpi.h>

class MPI
{
public:
  const static size_t MASTER_RANK = 0;

  MPI()
  {
    MPI_Init(nullptr, nullptr);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
  }
  ~MPI()
  {
    MPI_Finalize();
  }

  void send(void* buffer, int count, MPI_Datatype datatype, int dest)
  {
    MPI_Send(buffer, count, datatype, dest, 98, MPI_COMM_WORLD);
  }

  void send_to_others(void* buffer, int count, MPI_Datatype datatype)
  {
    for (int i = 0; i < size; ++i)
      if (i != rank)
```

```
        send(buffer, count, datatype, i);
  }

  void receive(void* buffer, int count, MPI_Datatype datatype, int source)
  {
    MPI_Recv(buffer, count, datatype, source, 98, MPI_COMM_WORLD, new
        MPI_Status);
  }

  static auto time()
  {
    return MPI_Wtime();
  }

  int get_size() const
  {
    return size;
  }
  int get_rank() const
  {
    return rank;
  }

private:
  int size;
  int rank;
};
```