МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение высшего образования  
 «Кемеровский государственный университет»**

**Институт фундаментальных наук**

**ДОМАШНЯЯ РАБОТА №6**

**ПО ДИСЦИПЛИНЕ**

**“Технологии параллельных вычислений”**

студента 3 курса

**Сулима Роман Иванович**

Направление 02.03.02 – Фундаментальная информатика и информационные технологии

Преподаватель:

к-т физ.-мат.наук, доцент

С.В. Стуколов

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работа защищена:

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_г.

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_

Кемерово 2021

СОДЕРЖАНИЕ

[1. Постановка задачи 2](#_Toc85837647)

[2. Описание используемых функций 2](#_Toc85837648)

[3. Реализация 2](#_Toc85837649)

[Задача 1. 2](#_Toc85837650)

[Задача 2. 4](#_Toc85837651)

[Заключение 7](#_Toc85837652)

[Литература 7](#_Toc85837653)

# 1. Постановка задачи

Выполните упражнения, приведенные в конце параграфа - 1-2 задачи. Оформляете в одном отчете - номер задачи, текст условия, текст программы с поясняющими комментариями в программе, скрин компиляции и скрин запуска на несколько процессов. Удачи!

# 2. Описание используемых функций

MPI\_Probe – Получить статус приходящего сообщения, до его прочтения.

MPI\_Get\_count – Получение размера сообщения, до его прочтения.

# 3. Реализация

## Задача 1.

Напишите программу, аналогичную из примера 4, но на каждом   
ненулевом процессе сформируйте одномерный массив длины,   
полученной датчиком случайных чисел от 1 до size. Для проверки   
корректности работы программы выведите отправляемые массивы на   
ненулевых процессах и полученный результат на 0-м процессе.

Программный код:

#include <stdio.h>

#include "mpi.h"

#include "stdlib.h"

#include "iostream"

#include <time.h>

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

    int rank;

    int size, count;

    MPI\_Status stat;

    MPI\_Init(&argc, &argv);

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

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

    if (rank == 0)

    {

        int\*\* a = new int\*[size];

        a[0] = new int[size\*size];

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

            a[i] = a[i-1] + size;

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

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

                a[i][j] = 0;

            }

        }

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

        {

            MPI\_Probe(MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &stat);

            MPI\_Get\_count(&stat, MPI\_INT, &count);

            MPI\_Recv(a[stat.MPI\_SOURCE], count, MPI\_INT, stat.MPI\_SOURCE, 777, MPI\_COMM\_WORLD, &stat);

        }

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

        {

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

                printf("%d ", a[i][j]);

            }

            printf("\n");

        }

    }

    else

    {

        srand(time(NULL) % (rank+465));

        int z = rand() % size;

        int \* b = new int[z];

        printf("rank = %d, b =", rank);

        for (int i = 0; i < z; i++)

        {

            b[i] = rank;

            printf("%d", b[i]);

        }

        printf("\n");

        MPI\_Send(b, z, MPI\_INT, 0, 777, MPI\_COMM\_WORLD);

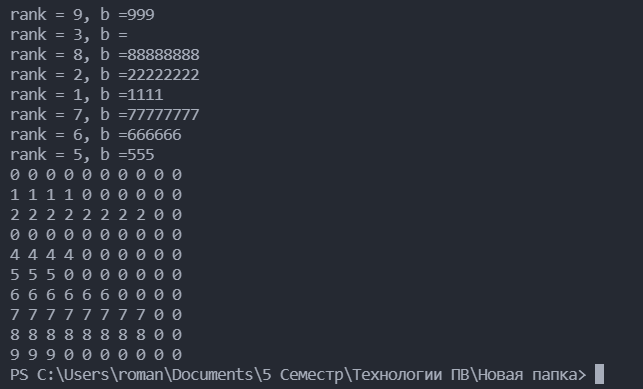
    }

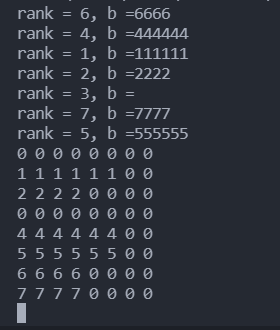
    MPI\_Finalize();

    return 0;

}

Выполнение:





## Задача 2.

Создайте последовательную программу (она уже должна быть выполнена   
в 1-й домашке), реализующую алгоритм суммирования ряда чисел для   
предела суммы n=(310 , 410...910 ). Член ряда равен (1/(1+i)). Создайте   
параллельную программу, реализующую данный алгоритм. Для   
реализации используйте изученные коммуникационные функции, а также   
возможности приема данных от неизвестного отправителя   
(MPI\_ANY\_SOURCE). Определите время выполнения последовательной   
и параллельной программ в зависимости от предела суммы (запустите   
несколько раз и возьмите наилучшие результаты). Параллельную   
программу запустите на 2-х и 4-х процессах. Требуется представить   
полученные значения в табличном виде, привести скрин компиляции и   
запуска параллельной программы. Сравните время выполнения   
параллельной программы на большом количестве процессов с   
использованием предопределенной переменной MPI\_ANY\_SOURCE и   
без нее.

Программный код:

Последовательная программа:

#include "mpi.h"

#include "stdlib.h"

#include <stdio.h>

#include "math.h"

int main(int argc, char \*argv[])

{

    int rank;

    int size;

    double t1, t2, sum = 0;

    MPI\_Status stat;

    MPI\_Init(&argc, &argv);

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

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

    for (int j = 3; j < 10; j++)

    {

        sum = 0;

        t1 = MPI\_Wtime();

        for (int i = 0; i < pow(10, j); i++)

        {

            sum += 1. / (i + 1);

        }

        t2 = MPI\_Wtime();

        printf("Predel 10^%d, sum = %g, time = %g\n",j,  sum, t2-t1);

    }

    MPI\_Finalize();

    return 0;

}

Параллельная программа.

#include "mpi.h"

#include "stdio.h"

#include "stdlib.h"

#include <math.h>

int main(int argc, char \*argv[])

{

    int n = 10;

    int rank, count, size;

    MPI\_Status stat;

    MPI\_Init(&argc, &argv);

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

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

    double start, end, sum1 = 0, sum2 = 0;

    for (int time = 3; time < 10; time++)

    {

        start = MPI\_Wtime();

        if (rank == 0)

        {

            double partial\_sum = 0;

            sum2 = 0;

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

            {

                partial\_sum = 0;

                MPI\_Recv(&partial\_sum, 1, MPI\_DOUBLE, i, 777, MPI\_COMM\_WORLD, &stat);

                sum1 += partial\_sum;

            }

            end = MPI\_Wtime();

            printf("Predel 10^%d, Summ = : %g. Exec time: %g\n", time, sum1, end - start);

        }

        else {

            sum2 = 0;

            for (int i = rank; i < pow(10, time); i += size)

            {

                sum2 += 1. / (i + 1);

            }

            MPI\_Send(&sum2, 1, MPI\_DOUBLE, 0, 777, MPI\_COMM\_WORLD);

        }

    }

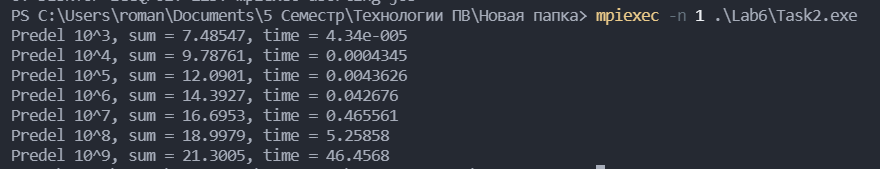
    MPI\_Finalize();

    return 0;

}

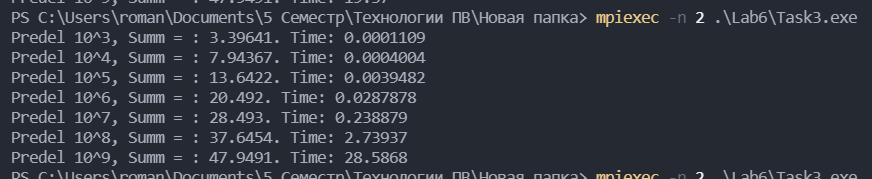
Выполнение:

Последовательная программа.

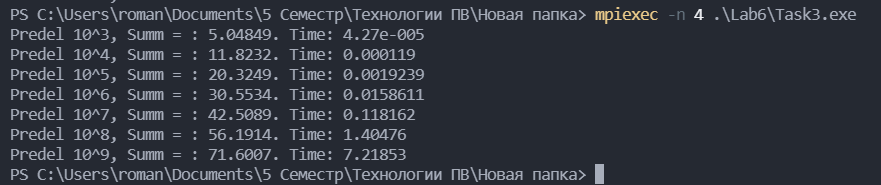


Параллельная программа.

На 2-х потоках.

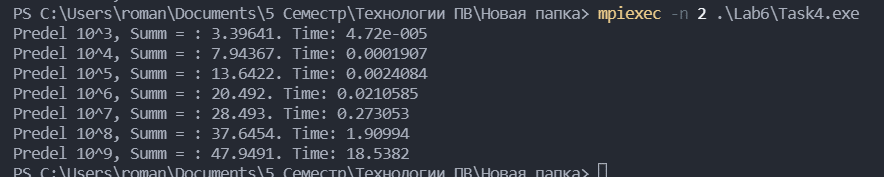


На 4-х потоках.

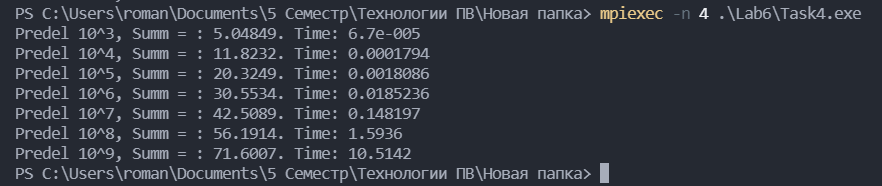


С использованием MPI\_ANY\_SOURCE.

На 2-х потоках:



На 4-х потоках.



# Заключение

# Литература