ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

**Отчет по лабораторной работе 4**

**по дисциплине: «Модели параллельного программирования»**

**Тема работы «Знакомство с технологией MPI»**

студента очного отделения

2 курса 12001801 группы

Капустина Виктора Сергеевича

Проверил(а):

Петров Денис Васильевич

Белгород 2020

**Теоретическая часть.**

Первой вызываемой функцией MPI должна быть функция:

*int MPI\_Init* ( *int \*agrc, char \*\*\*argv* ) для инициализации среды выполнения MPI-программы. Параметрами функции являются количество аргументов в командной строке и текст самой командной строки. Последней вызываемой функцией MPI обязательно должна являться функция: *int MPI\_Finalize (void).*

Определение количества процессов в выполняемой параллельной программе осуществляется при помощи функции: *int MPI\_Comm\_size ( MPI\_Comm comm, int \*size ).* Для определения ранга процесса используется функция: *int MPI\_Comm\_rank ( MPI\_Comm comm, int \*rank ).*

Достижение эффективного выполнения операции передачи данных от одного процесса всем процессам программы может быть обеспечено при помощи функции MPI: *int MPI\_Bcast(void \*buf,int count,MPI\_Datatype type,int root,MPI\_Comm comm)*, где - *buf, count, type* – буфер памяти с отправляемым сообщением (для процесса с рангом 0), и для приема сообщений для всех остальных процессов, - root - ранг процесса, выполняющего рассылку данных, - comm - коммуникатор, в рамках которого выполняется передача данных.

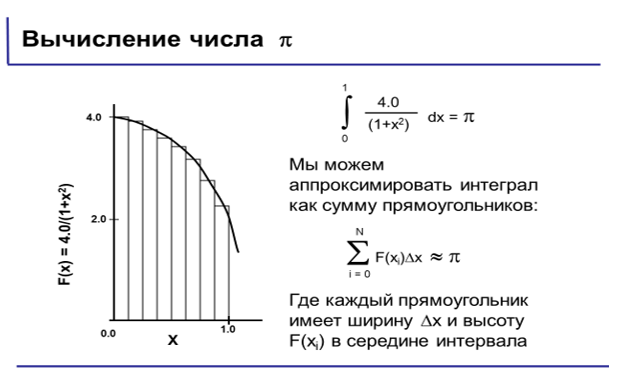
Для наилучшего выполнения действий, связанных с редукцией данных, в MPI предусмотрена функция: *int MPI\_Reduce(void \*sendbuf, void \*recvbuf,int count,MPI\_Datatype type, MPI\_Op op,int root,MPI\_Comm comm)*, где - *sendbuf* - буфер памяти с отправляемым сообщением, - *recvbuf* – буфер памяти для результирующего сообщения (только для процесса с рангом root), - *count* - количество элементов в сообщениях, - *type* – тип элементов сообщений, - *op* - операция, которая должна быть выполнена над данными, - *root* - ранг процесса, на котором должен быть получен результат, - *comm* - коммуникатор, в рамках которого выполняется операция.

**Ход работы**

**Цель:** Ознакомиться с технологией MPI. Научиться компилировать и запускать MPI программы. Получить навык работы с простейшими средствами передачи сообщений между процессами.

**Индивидуальное задание: 17 Вариант** (задание 2)Разработайте программу с использованием технологии MPI на ресурсе clusterknit.bsu.edu.ru, для вычисления значения функции f(x) на отрезке [1,N+1] с шагом h=N/P , где N – номер варианта и составьте таблицы №1 и №2.

****

**Задание 1.  
**

**Расчёт числа Pi на clusterknit.bsu.edu.ru**

**Листинг 1**

#include "mpi.h"

#include <stdio.h>

#include <math.h>

double f(double a)

{

return (4.0 / (1.0 + a\*a));

}

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

{

int done = 0, n, myid, numprocs, i;

double PI25DT = 3.141592653589793238462643;

double mypi, pi, h, sum, x;

double startwtime = 0.0, endwtime;

int namelen;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

MPI\_Init(&argc,&argv);

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

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

MPI\_Get\_processor\_name(processor\_name,&namelen);

fprintf(stdout,"Process %d of %d is on %s\n",

myid, numprocs, processor\_name);

fflush(stdout);

while (!done) {

if (myid == 0) {

fprintf(stdout, "Enter the number of intervals: (0 quits) ");

fflush(stdout);

if (scanf("%d",&n) != 1) {

fprintf( stdout, "No number entered; quitting\n" );

n = 0;

}

startwtime = MPI\_Wtime();

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (n == 0)

done = 1;

else {

h = 1.0 / (double) n;

sum = 0.0;

for (i = myid + 1; i <= n; i += numprocs) {

x = h \* ((double)i - 0.5);

sum += f(x);

}

mypi = h \* sum;

MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (myid == 0) {

printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));

endwtime = MPI\_Wtime();

printf("wall clock time = %f\n", endwtime-startwtime);

fflush( stdout );

}

}

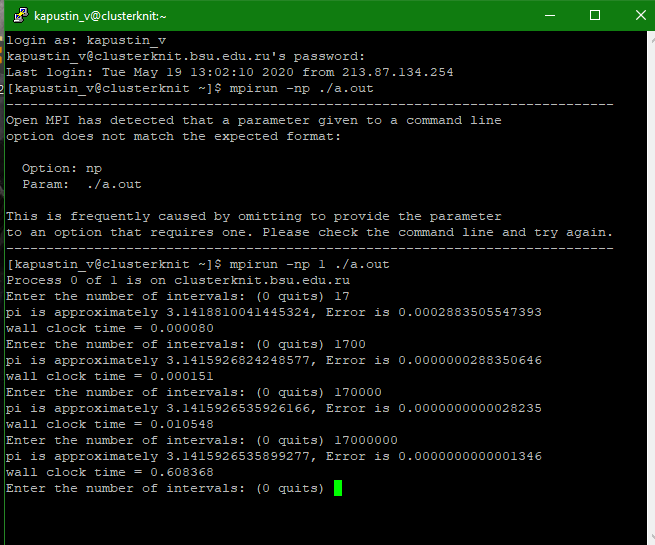
}

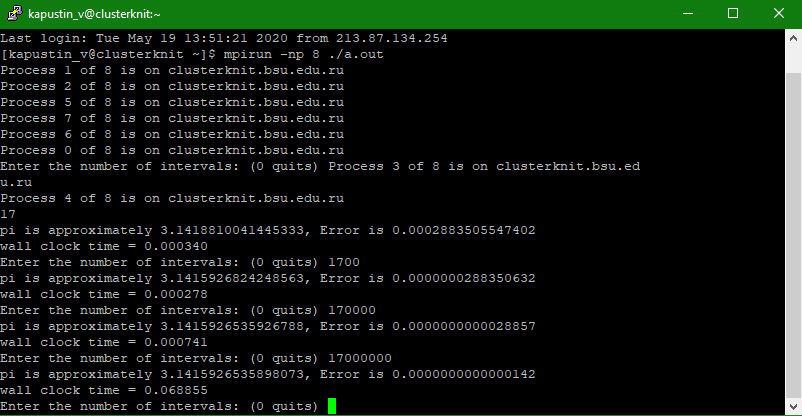
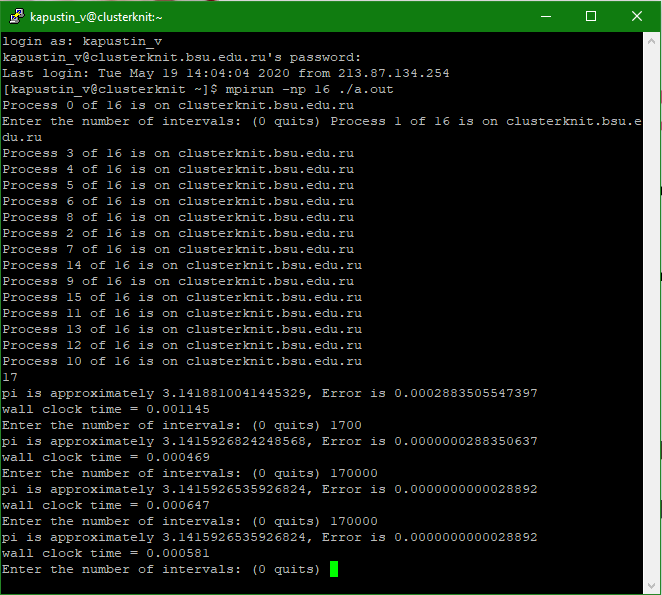
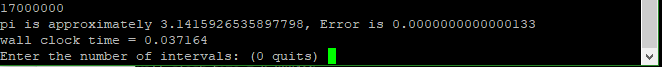
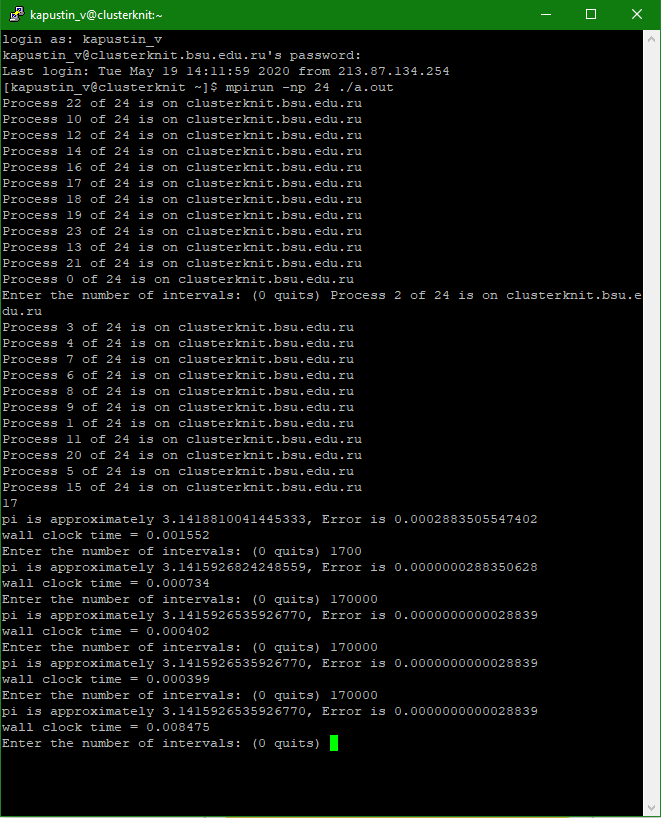
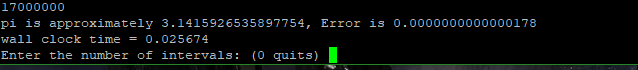
MPI\_Finalize();

return 0;

}

**Конец листинга 1**

**  
Скриншот работы программы на 1 узле**

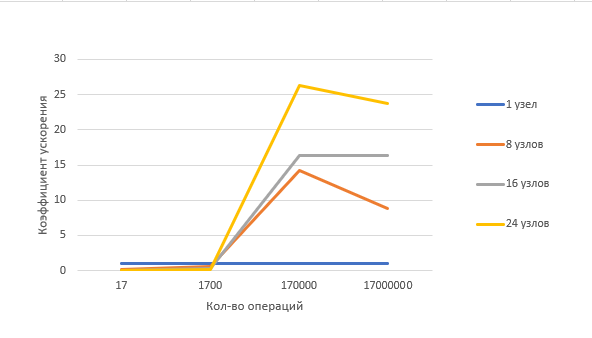
  
**Скриншот работы программы на 8 узлах**  
  
  
**Скриншоты работы программы на 16 узлах  
  
  
  
Скриншоты работы программы на 24 узлах**

**Таблица №1. Расчет числа π.**

| Номер  расчета | Количество  итераций | на 1 узле | | на 8 узлах | | на 16 узлах | | на 24 узлах | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Значение числа π | Время расчета | Значение числа π | Время расчета | Значение числа π | Время расчета | Значение числа π | Время расчета |
| 1 | 17 | 3.1418810041445324 | 0.000080 | 3,141881004144533 | 0.000340 | 3.1418810041445329 | 0.001145 | 3.1418810041445333 | 0.001552 |
| 2 | 1700 | 3.1415926824248577 | 0.000151 | 3,1415926824248563 | 0.000278 | 3.1415926824248568 | 0.000469 | 3.1415926824248559 | 0.000731 |
| 3 | 170000 | 3.1415926535926166 | 0.010548 | 3.1415926659172100 | 0.000741 | 3.1415926535926824 | 0.000647 | 3.1415926535926770 | 0.000402 |
| 4 | 17000000 | 3.1415926537130789 | 0.608368 | 3.1415926537130714 | 0.068855 | 3.1415926535897798 | 0.037164 | 3.1415926535897754 | 0.025674 |

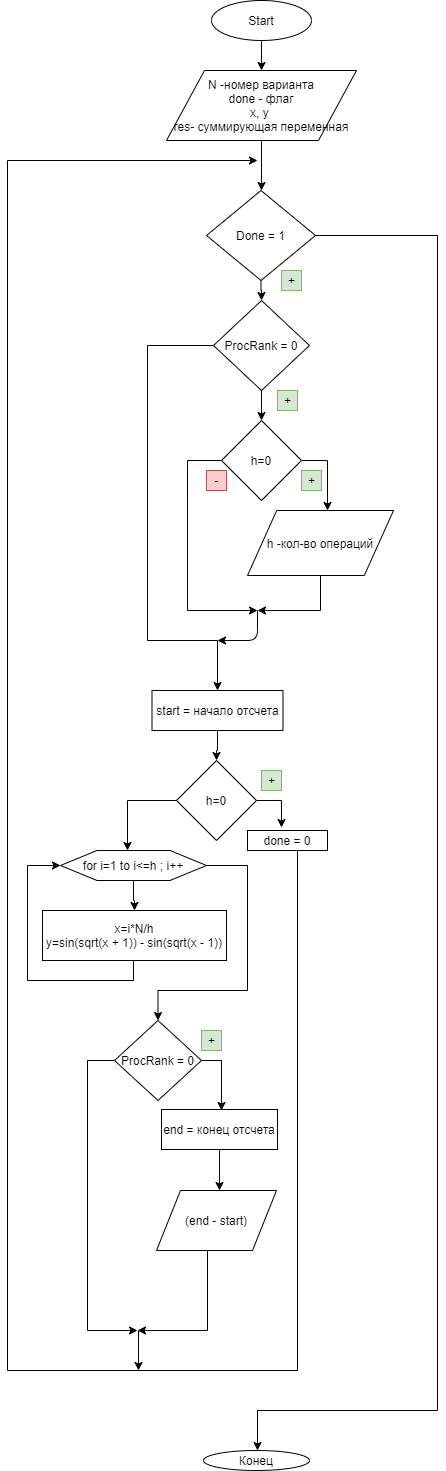
**Таблица №2. Коэффициент ускорения ( Кi= Ti1/ Ti j, где i – номер расчета, j - кол-во узлов)**

| № | P – количество  расчетов | Коэффициент  ускорения К на 1 узле | Коэффициент  ускорения К на 8 узлах | Коэффициент  ускорения К на 16 узлах | Коэффициент  ускорения К на 24 узлах |
| --- | --- | --- | --- | --- | --- |
| 1 | 17 | 1 | 0,2352 | 0,0698 | 0,0515 |
| 2 | 1700 | 1 | 0,5431 | 0,3219 | 0,2117 |
| 3 | 170000 | 1 | 14,2348 | 16,3029 | 26,2388 |
| 4 | 17000000 | 1 | 8,8354 | 16,3698 | 23,6958 |

  
**График 1. График зависимости коэффициента ускорения от количества расчётов в clusterknit.bsu.edu.ru.**

**Задание 2**

**Разработайте программу с использованием технологии MPI на ресурсе clusterknit.bsu.edu.ru, для вычисления значения функции f(x) на отрезке [1,N+1] с шагом h=N/P , где N – номер варианта**

**Блок-схема  
**

**Листинг 2**

#include<iostream>

#include<stdlib.h>

#include<iomanip>

#include<math.h>

#include<cmath>

#include"mpi.h"

using namespace std;

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

setlocale(LC\_ALL, "RUS");

const int N = 17;

int ProcNum, ProcRank;

double start, end;

bool done = false;

double res = 0;

double h;

double x = 0;

double y = 0;

MPI\_Init(&argc, &argv); //инициализация начала MPI-программы

MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);//функция расчёта количества процессоров

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);//функция расчёта ранга процессоров

while (!done) {

if (ProcRank == 0) {

printf("Input interval: (0 - exit) ");

cin >> h;

if (&h == 0) {

printf("Program exit\n");

}

start = MPI\_Wtime();//начало отсчёта времени

}

MPI\_Bcast(&h, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);//отправка сообщения от одного процесса программы всем процессам программы

if (h == 0)

done = true;

else {

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

x = i \* (N / h);

y = sin(sqrt(x + 1)) - sin(sqrt(x - 1));

}

MPI\_Reduce(&y, &res, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);//отправка сообщения от всех процессов программы одному процессу программы

if (ProcRank == 0) {

end = MPI\_Wtime();//конец отсчёта времени

cout << "Time of working program: " << (end - start) \* 1000 << endl;

cout << setprecision(7) << res << endl;

}

}

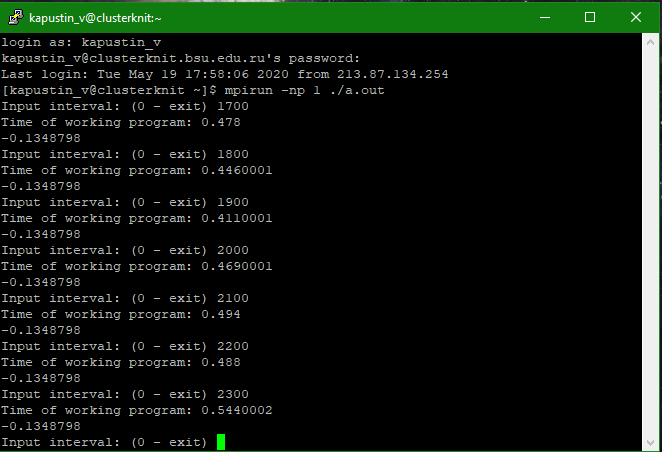
}

MPI\_Finalize();//Конец MPI-программы

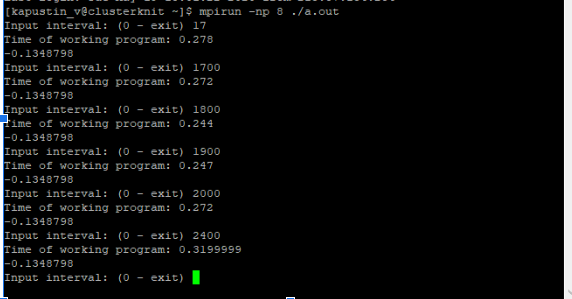
}

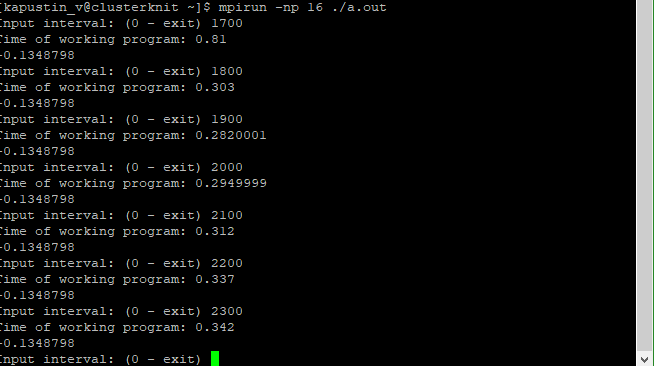
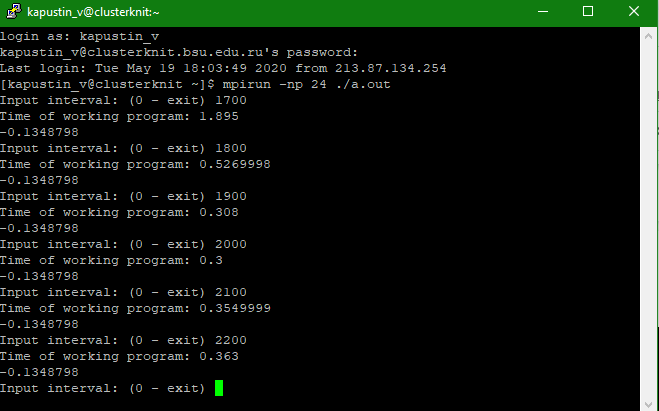
**Конец листинга 2**

**Скриншоты работы программы в тестовом режиме.**

****

**Скриншот работы программы на 1 узле**

**  
Скриншот работы программы на 8 узлах**

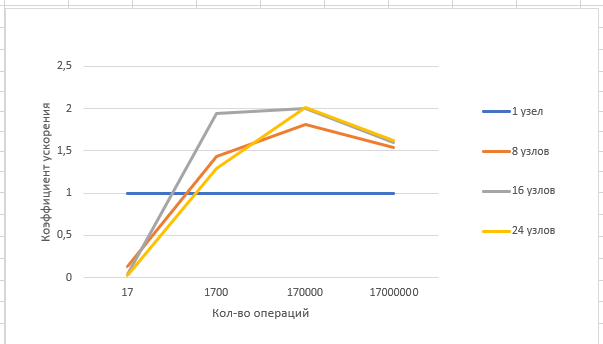
**  
Скриншот работы программы на 16 узлах  
  
Скриншот работы программы на 24 узлах**

**Таблица №1. Время расчета значений функции**

| Номер  расчета | Количество  итераций | Время расчета на 1 узле, ms | Время расчета на 8 узлах, ms | Время расчета на 16 узлах, ms | Время расчета на 24 узлах, ms |
| --- | --- | --- | --- | --- | --- |
| 1 | 17 | 0.038 | 0.287 | 0.93 | 1.475 |
| 2 | 1700 | 0.4099 | 0.2850 | 0.211 | 0,318 |
| 3 | 170000 | 35,189 | 19,443 | 17,546 | 17,445 |
| 4 | 17000000 | 2959.818 | 1927.628 | 1854,648 | 1824,297 |

**Таблица №2. Коэффициент ускорения ( Кi= Ti1/ Ti j, где i – номер расчета, j - кол-во узлов)**

| № | P – количество  расчетов | Коэффициент  ускорения К на 1 узле | Коэффициент  ускорения К на 8 узлах | Коэффициент  ускорения К на 16 узлах | Коэффициент  ускорения К на 24 узлах |
| --- | --- | --- | --- | --- | --- |
| 1 | 17 | 1 | 0,1324 | 0,0408 | 0,0257 |
| 2 | 1700 | 1 | 1,4382 | 1,9426 | 1,2889 |
| 3 | 170000 | 1 | 1,8098 | 2,0055 | 2,0171 |
| 4 | 17000000 | 1 | 1,5354 | 1,5958 | 1,6224 |

**  
График 2. График зависимости коэффициента ускорения от количества расчётов в clusterknit.bsu.edu.ru.**

**Вывод:** По итогам проведенной работы можно удостовериться, что распараллеливание программы действительно повышает скорость её работы. Это можно увидеть как в программе по вычислению числа Pi, так и в программе по расчёту функции. Однако стоит учитывать, что использование парной операции передачи данных с помощью цикла for приводит к латентности.