МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Параллельные Вычисления»

Тема: «Передача данных между процессами»

Студент гр. 1307 _____ Тростин М. Ю.

Преподаватель _____

Санкт-Петербург

2025

Цели и задачи
Освоить функции передачи данных между процессами

1) Запустить 4 процесса. На каждом процессе создать переменные: ai,bi,ci, где I – номер процесса. Инициализировать переменные. Вывести данные на печать. Передать данные на другой процесс. Напечатать номера процессов и поступившие данные. Найти:
   - c0=a1+b2;
   - c1=a3+b0;
   - c2=a0+b3;
   - c3=a2+b1.
2) Запустить n процессов и найти среднее арифметическое элементов вектора;

Задание 1

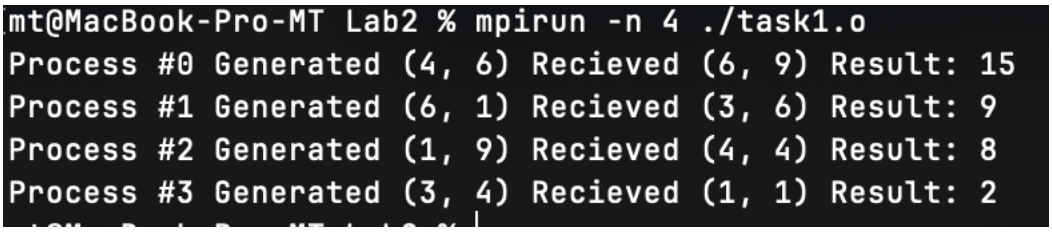
Запустить 4 процесса. На каждом процессе создать переменные: ai,bi,ci, где I – номер процесса. Инициализировать переменные. Вывести данные на печать. Передать данные на другой процесс. Напечатать номера процессов и поступившие данные. Найти:

- c0=a1+b2;
- c1=a3+b0;
- c2=a0+b3;
- c3=a2+b1.

Каждый из процессов генерирует случайные числа a и b, после чего определяет каким из процессам необходимо отправить a, b, а также от каких процессов получить a, b в зависимости от своего ранка.

Исходный код доступен в Приложении А

Скриншот выполнения



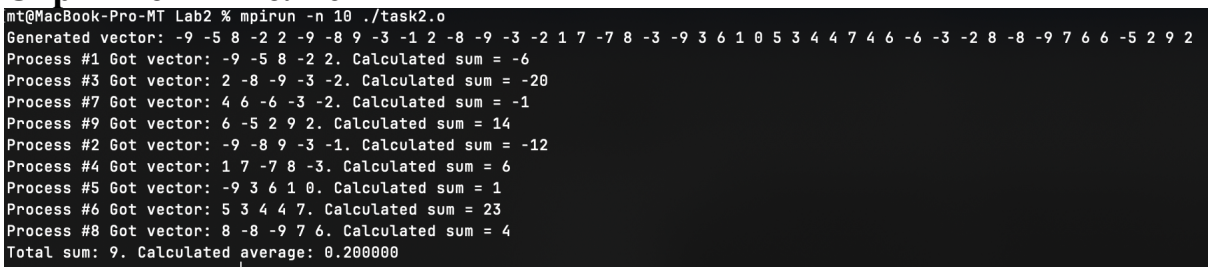
Задание 2
Запустить n процессов и найти среднее арифметическое элементов вектора

В данном задании «главный процесс» (с rank = 0) в наяале генеириует массив случайных чисел. Размер вектора зависит от количества запущенных процессов: на каждый другой процесс генерируется несколько чисел. После генерации, главный процесс отправляет каждому другому процессу соответствующую часть массива.

Получив данные, процессы считают сумму своей части и отправляют ответ назад главному процессу. Получив и сложив суммы частей массива, главный процесс может найти среднее арифметическое всего массива, разделив общую сумму на кол-во сгенерированных элментов

Исходный код доступен в Приложении Б

Скриншот выполнения



Выводы

В рамках выполнения данной работы были освоены функции передачи данных между процессами

ПРИЛОЖЕНИЕ А

Задание 1. Файл task1.cpp

```
#include <stdio.h>
#include <mpi.h>
#include <random>

#define rid_t int
#define COMM_SIZE 4

typedef struct {
    rid_t rank;

    rid_t aReceiver;
    rid_t bReceiver;
    rid_t aSender;
    rid_t bSender;

    int aGenerated;
    int bGenerated;
    int a;
    int b;
} MailInfo;

MailInfo getMailInfo(int rank) {
    MailInfo mi;
    mi.rank = rank;
    mi.a = -1;
    mi.b = -1;

    switch (rank) {
    case 0:
        mi.aReceiver = 2;
        mi.bReceiver = 1;
        mi.aSender = 1;
        mi.bSender = 2;
        break;
    case 1:
        mi.aReceiver = 0;
        mi.bReceiver = 3;
        mi.aSender = 3;
        mi.bSender = 0;
        break;
    case 2:
        mi.aReceiver = 3;
        mi.bReceiver = 0;
        mi.aSender = 0;
        mi.bSender = 3;
        break;
    case 3:
        mi.aReceiver = 1;
```

```
        mi.bReceiver = 2;
        mi.aSender = 2;
        mi.bSender = 1;
        break;
    }

    srand((unsigned int)(time(NULL) * (rank + 1)));
    mi.aGenerated = (rand() % 9) + 1;
    mi.bGenerated = (rand() % 9) + 1;

    return mi;
}

void serveRank(MailInfo * mi, int client) {
    if (client == mi->rank) {
        MPI_Send(&mi->aGenerated, 1, MPI_INT, mi->aReceiver, mi->rank *
10 + mi->aReceiver, MPI_COMM_WORLD);
        MPI_Send(&mi->bGenerated, 1, MPI_INT, mi->bReceiver, mi->rank *
10 + mi->bReceiver, MPI_COMM_WORLD);
        return;
    }

    if (client == mi->aSender) {
        MPI_Status s;
        MPI_Recv(&mi->a, 1, MPI_INT, mi->aSender, mi->aSender * 10 +
mi->rank, MPI_COMM_WORLD, &s);
        return;
    }

    if (client == mi->bSender) {
        MPI_Status s;
        MPI_Recv(&mi->b, 1, MPI_INT, mi->bSender, mi->bSender * 10 +
mi->rank, MPI_COMM_WORLD, &s);
        return;
    }
}

int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size != COMM_SIZE) {
        if (rank == 0) printf("Expected Comm_size: %d. Got: %d\n",
COMM_SIZE, size);
        MPI_Finalize();
        return EINVAL;
    }
```

```
    MailInfo mi = getMailInfo(rank);
    for (int i = 0; i < 4; i++) serveRank(&mi, i);
    printf("Process #%d Generated (%d, %d) Recieved (%d, %d) Result:
%d\n", mi.rank, mi.aGenerated, mi.bGenerated, mi.a, mi.b, mi.a + mi.b);

    MPI_Finalize();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

Задание 2. Файл task2.cpp

```
#include <stdio.h>
#include <mpi.h>
#include <random>
#include <math.h>

#define vector_t int *
#define element_t int

#define SLICE_SIZE 5

int getSum(vector_t v) {
    int sum = 0;
    for (int i = 0; i < SLICE_SIZE; i++) sum += v[i];
    return sum;
}

vector_t getRandomVector(size_t length) {
    vector_t vector = (vector_t)malloc(length * sizeof(element_t));
    srand((unsigned int)time(NULL));
    for (int i = 0; i < length; i++) vector[i] = rand() % 19 - 9;
    return vector;
}

int mainProcess(int size) {
    size_t vectorLength = (size - 1) * SLICE_SIZE;
    vector_t vector = getRandomVector(vectorLength);

    printf("Generated vector:");
    for (int i = 0; i < vectorLength; i++) printf(" %d", vector[i]);
    printf("\n");

    for (int i = 1; i < size; i++) {
        vector_t vptr = vector + (i - 1) * SLICE_SIZE;
        int errc = MPI_Send(vptr, SLICE_SIZE, MPI_INT, i, i,
MPI_COMM_WORLD);
        if (errc != MPI_SUCCESS) return errc;
    }

    int sum = 0;
    for (int i = 1; i < size; i++) {
        MPI_Status s;
        int localResult;
        int errc = MPI_Recv(&localResult, 1, MPI_INT, i, size + i,
MPI_COMM_WORLD, &s);
        if (errc != MPI_SUCCESS) return errc;
        sum += localResult;
    }
```

```
    float avg = (float)(sum) / vectorLength;
    printf("Total sum: %d. Calculated average: %f\n", sum, avg);
    free(vector);

    return MPI_SUCCESS;
}

int calculatorProcess(int rank, int size) {
    element_t v[SLICE_SIZE];
    MPI_Status s;
    int errc = MPI_Recv(&v, SLICE_SIZE, MPI_INT, 0, rank,
MPI_COMM_WORLD, &s);
    if (errc != MPI_SUCCESS) return errc;

    int sum = getSum(v);
    printf("Process #%d Got vector: %d %d %d %d %d. Calculated sum =
%d\n", rank, v[0], v[1], v[2], v[3], v[4], sum);

    return MPI_Send(&sum, 1, MPI_INT, 0, size + rank, MPI_COMM_WORLD);
}

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size < 2) {
        if (rank == 0) printf("Expected more than one process\n");
        MPI_Finalize();
        return EINVAL;
    }

    int errc = !rank ? mainProcess(size) : calculatorProcess(rank,
size);

    MPI_Finalize();
    return errc;
}
```