

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего профессионального образования**

**«Ивановский государственный энергетический университет**

**имени В.И. Ленина»**

Кафедра программного обеспечения компьютерных систем

## **Отчет по лабораторным работам**

Дисциплина: Технологии параллельного программирования

Выполнил:

студент группы 3-41хх

Прохоров М. В.

Проверила:

Чернышева Л.П.

Иваново, 2024 г.

## Тема 1: «Технология параллельного программирования MPI»

**Задание 1:** Набрать программу: каждый процесс определяет общее число процессов `size` и свой собственный уникальный номер `rank` и записывает эти значения в свой файл. Выполнить программу. Проверить результаты. Подготовить отчет.

**Цель:** познакомиться с возможностями MPI.

### Код программы:

```
#include <iostream>
#include <mpi.h>
#include <cstdio>
#include "lab1.h"
using namespace std;

int lab1 (int size, int rank) {
    FILE* f;
    char name[20];
    sprintf(name, "rank_%d.txt", rank);

    if ((f = fopen(name, "w")) == nullptr) {
        fprintf(stderr, "File error\n");
        MPI_Finalize();
        return 4;
    }

    fprintf(f, "size=%d rank=%d\n", size, rank);
    fclose(f);

    return 0;
}

int main(int argc, char** argv) {
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) return 1;

    int size, rank;

    if (MPI_Comm_size(MPI_COMM_WORLD, &size) != MPI_SUCCESS) {
        MPI_Finalize();
        return 2;
    }

    if (MPI_Comm_rank(MPI_COMM_WORLD, &rank) != MPI_SUCCESS) {
        MPI_Finalize();
        return 3;
    }

    MPI_Finalize();
    return 0;
}
```

### Результат программы:

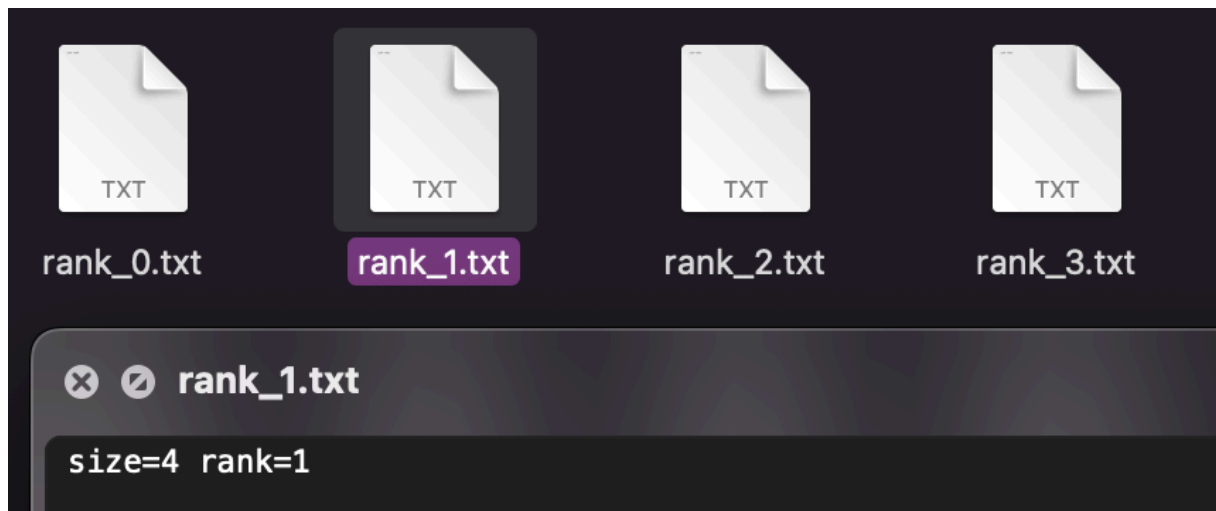


Рис. 1. Результат выполнения программы

**Вывод:** В результате выполнения данной лабораторной работы мы научились пользоваться базовыми возможностями работы с MPI.

**Задание 2:** Написать программу: все процессы последовательно записывают в один файл общее число процессов `size` и свой собственный уникальный номер `rank`. Использовать функцию `MPI_Barrier()`. Выполнить программу. Проверить результат. Добавить в отчет.

**Цель:** познакомиться с функцией `int MPI_Barrier(MPI_Comm comm);`.

#### Код программы:

```
#include "lab2.h"

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

int lab2(int size, int rank) {
    if (rank == 0) {
        FILE* file = fopen("output.txt", "w");
        if (file == NULL) {
            printf("Ошибка при открытии файла\n");
            MPI_Abort(MPI_COMM_WORLD, 1);
        }
        fprintf(file, "Общее количество процессов: %d\n", size);
        fclose(file);
    }

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

        if (i == rank) {
            FILE* file = fopen("output.txt", "a");
            if (file == NULL) {
                printf("Ошибка при открытии файла\n");
                MPI_Abort(MPI_COMM_WORLD, 1);
            }
            fprintf(file, "Процесс %d из %d\n", rank, size);
            fclose(file);
        }
        MPI_Barrier(MPI_COMM_WORLD);
    }
}
```

```

    return 0;
}
int main(int argc, char** argv) {
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) return 1;

    int size, rank;

    if (MPI_Comm_size(MPI_COMM_WORLD, &size) != MPI_SUCCESS) {
        MPI_Finalize();
        return 2;
    }

    if (MPI_Comm_rank(MPI_COMM_WORLD, &rank) != MPI_SUCCESS) {
        MPI_Finalize();
        return 3;
    }

    MPI_Barrier(MPI_COMM_WORLD);

    lab2(size, rank);

    MPI_Finalize();
    return 0;
}

```

### Результат программы:

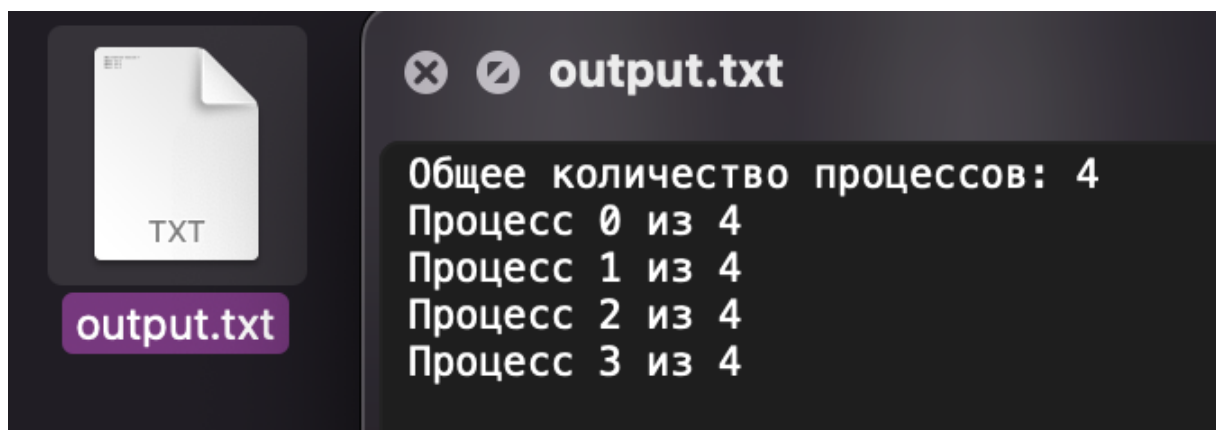


Рис.2. Результат

**Вывод:** В ходе выполнения данной лабораторной работы мы познакомились с возможностями MPI, использовали функцию `MPI_Barrier`, которая создает барьер синхронизации для группы процессов, смысл которого заключается в следующем: каждый процесс при достижении барьера блокируется, пока все процессы из группы не достигнут барьера.

**Задание 3:** Часть 1. Написать программу: нулевой процесс получает номера процессов от всех остальных процессов, собирает эти значения в массив. Затем рассылает этот массив на все остальные процессы. Остальные процессы пересылают на нулевой процесс свой rank, затем получают массив и его записывают в свой файл. Реализовать с помощью функций MPI\_Send() и MPI\_Recv().

Часть 2. Сделать то же, используя функцию MPI\_Bcast(). Выполнить программу. Проверить результаты. Добавить в отчет.

**Цель:** познакомиться с функциями для работы с сообщениями:

int MPI\_Send(const void \*buf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm);

int MPI\_Recv(void \*buf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status);

int MPI\_Bcast(void \*buffer, int count, MPI\_Datatype datatype, int root, MPI\_Comm comm);

**Код программы:**

```
#include "lab3.h"

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

void useRecv(int rank, int size)
{
    MPI_Status status;
    int* a = new int[size];
    a[rank] = rank;
    if (!rank) // если нулевой процесс
    {
        for (int i = 1; i < size; i++)
        {
            MPI_Recv(&a[i], 1, MPI_INT, i, MPI_ANY_TAG, MPI_COMM_WORLD,
&status);
        }

        for (int i = 1; i < size; i++)
        {
            MPI_Send(a, size, MPI_INT, i, 1, MPI_COMM_WORLD);
        }
    }
    else
    {
        MPI_Send(&a[rank], 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
        MPI_Recv(a, size, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    }

    char name[20];
    sprintf(name, "rank_recv_%d.txt", rank);
    FILE* f = fopen(name, "w");
    if (f != nullptr) {
        for (int i = 0; i < size; i++)
        {
            fprintf(f, "rank: %d \n", a[i]);
        }
        fclose(f);
    }
    delete[] a;
}
```

```

}

void useBcast(int rank, int size)
{
    int* a = new int[size];
    a[rank] = rank;

    for (int i = 0; i < size; i++)
    {
        MPI_Bcast(&a[i], 1, MPI_INT, i, MPI_COMM_WORLD);
    }

    char name[20];
    sprintf(name, "rank_bcast_%d.txt", rank);
    FILE* f = fopen(name, "w");
    if (f != nullptr) {
        for (int i = 0; i < size; i++)
        {
            fprintf(f, "rank: %d \n", a[i]);
        }
        fclose(f);
    }
    delete[] a;
}

int main(int argc, char** argv) {
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) return 1;

    int size, rank;

    if (MPI_Comm_size(MPI_COMM_WORLD, &size) != MPI_SUCCESS) {
        MPI_Finalize();
        return 2;
    }

    if (MPI_Comm_rank(MPI_COMM_WORLD, &rank) != MPI_SUCCESS) {
        MPI_Finalize();
        return 3;
    }

    useRecv(rank, size);
    useBcast(rank, size);
    MPI_Finalize();
    return 0;
}

```

### Результат программы 3:

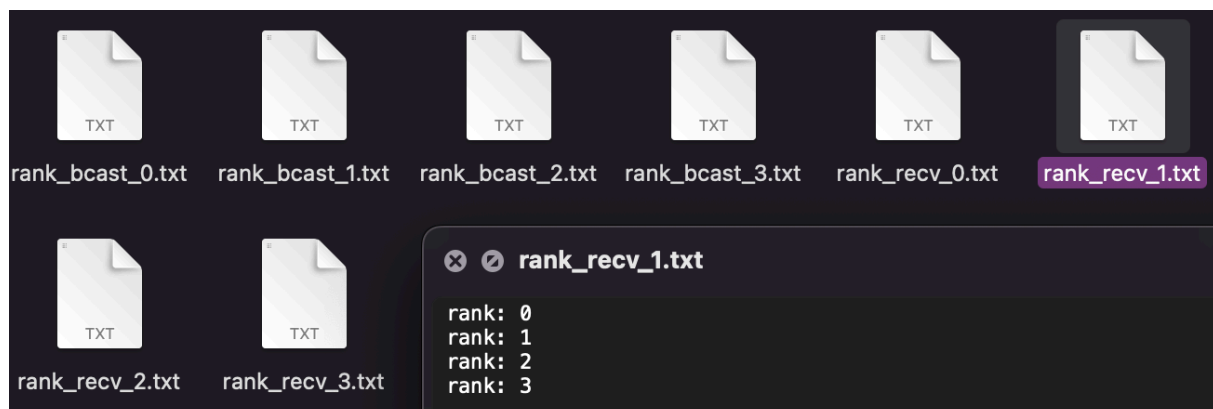


Рис.3. Результат

**Вывод:** В ходе выполнения данной лабораторной работы мы познакомились с функциями для работы с сообщениями MPI\_Send(), MPI\_Recv(), MPI\_Bcast(). Сравнивая 2 метода реализации, можно заметить, что первая программа намного объемнее и сложнее второй, поэтому предпочтительнее использовать именно второй вариант (MPI\_Bcast()).

**Задание IV.** Написать программу: на каждом процессе создается массив, причем на разных процессах длина массива различна. Собрать эти массивы на одном процессе с номером root в один большой массив. Массивы должны быть записаны последовательно. Использовать функции MPI\_GathervMPI\_Gatherv() и MPI\_Gatherv(). Добавить в отчет.

**Цель:** познакомиться с функциями для работы с сообщениями:

int MPI\_Allgather(const void \*sendbuf, int sendcount, MPI\_Datatype sendtype,

void \*recvbuf, int recvcount, MPI\_Datatype recvtype, MPI\_Comm comm);

int MPI\_Gatherv(const void \*sendbuf, int sendcount, MPI\_Datatype sendtype, void \*recvbuf, const int \*recvcounts, const int \*displs, MPI\_Datatype recvtype, int root, MPI\_Comm comm);

**Код программы:**

```
void lab4(int rank, int size, const int root)
{
    srand(time(NULL) + rank);
    int number = rand() % 10 + 1;

    int* arr = new int[number];
    for (int i = 0; i < number; i++)
    {
        arr[i] = rank * 10 + i;
    }

    int* countArray = new int[size];
    MPI_Allgather(&number, 1, MPI_INT, countArray, 1, MPI_INT,
MPI_COMM_WORLD);
    int* displacement = new int[size] { 0 };

    for (int i = 1; i < size; ++i)
    {
        displacement[i] = displacement[i - 1] + countArray[i - 1];
    }

    int resultArraySize = 0;
    for (int i = 0; i < size; ++i)
    {
        resultArraySize += countArray[i];
    }

    int* resultArray = new int[resultArraySize];
    for (int i = 0; i < resultArraySize; ++i)
    {
        resultArray[i] = 0;
    }
}
```

```

    }

    MPI_Gatherv(arr, number, MPI_INT, resultArray, countArray, displacement,
MPI_INT, root, MPI_COMM_WORLD);

    if (rank == root)
    {
        char name[20];
        sprintf(name, "result.txt");
        FILE* f = fopen(name, "w");
        if (f != NULL)
        {
            for (int i = 0; i < resultArraySize; ++i)
            {
                fprintf(f, "b[%d]: %d \n", i, resultArray[i]);
            }
            fclose(f);
        }
    }

    MPI_Finalize();
    delete[] arr;
    delete[] countArray;
    delete[] displacement;
    delete[] resultArray;
}

```

**Результат программы 4:**



```
result.txt ×  
1 b[0]: 0  
2 b[1]: 1  
3 b[2]: 2  
4 b[3]: 3  
5 b[4]: 10  
6 b[5]: 20  
7 b[6]: 21  
8 b[7]: 22  
9 b[8]: 23  
10 b[9]: 24  
11 b[10]: 25  
12 b[11]: 26  
13 b[12]: 27  
14 b[13]: 30  
15 b[14]: 31  
16 b[15]: 32  
17 b[16]: 33  
18 b[17]: 34  
19
```

**Вывод:** В результате работы мы познакомились с тем, как можно передавать данные между процессами с помощью функций `MPI_Gatherv` и `MPI_Allgatherv`.

**Задание V.** Пусть матрица  $a[n][m]$  хранится на процессе с  $\text{rank} = 0$ . Пусть дано четыре процесса. Процесс 0 посылает части этой матрицы другим процессам и себе тоже. Процесс 0 получает  $a[i][j]$  для  $i=1, \dots, n/2$  и  $j=1, \dots, m/2$ . Процесс 1 получает  $a[i][j]$  для  $i=n/2+1, \dots, n$  и  $j=1, \dots, m/2$ . Процесс 2 получает  $a[i][j]$  для  $i=1, \dots, n/2$  и  $j=m/2+1, \dots, m$ . Процесс 3 получает  $a[i][j]$  для  $i=n/2+1, \dots, n$  и  $j=m/2+1, \dots, m$ . Это двумерная декомпозиция матрицы  $a[n][m]$  на четыре процесса. Написать программу рассылки частей матрицы по процессам. Использовать функцию `MPI_Scatterv()`. Добавить в отчет.

**Цель:** познакомиться с функциями для работы с сообщениями:

`int MPI_Scatterv(const void *sendbuf, const int *sendcounts, const int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);`

**Код программы:**

```
int lab5(int rank, int size, const int root = 0)
{
    char name[20];
    sprintf(name, "result5.txt");
    FILE* f = fopen(name, "w");

    const int n = 8;
    const int m = 10;

    int arr[n][m];
    int bArr[n / 2][m / 2];
    int* count = new int[size];
    int* displacement = new int[size];

    if (rank == root)
    {
        for (int i = 0; i < n / 2; ++i)
        {
            for (int j = 0; j < m / 2; ++j)
            {
                arr[i][j] = 0;
            }
        }
        for (int i = n / 2; i < n; ++i)
        {
            for (int j = 0; j < m / 2; ++j)
            {
                arr[i][j] = 1;
            }
        }
        for (int i = 0; i < n / 2; ++i)
        {
            for (int j = m / 2; j < m; ++j)
            {
                arr[i][j] = 2;
            }
        }
        for (int i = n / 2; i < n; ++i)
        {
            for (int j = m / 2; j < m; ++j)
```

```

        {
            arr[i][j] = 3;
        }
    }
    if (f != nullptr)
    {
        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < m; ++j)
            {
                fprintf(f, "%d", arr[i][j]);
            }
            fprintf(f, "\n");
        }
        fprintf(f, "\n");
        fclose(f);
    }
}

for (int i = 0; i < size; ++i)
{
    count[i] = m / 2;
}

displacement[0] = 0;
displacement[1] = n / 2 * m;
displacement[2] = m / 2;
displacement[3] = n / 2 * m + m / 2;

for (int i = 0; i < n / 2; ++i)
{
    MPI_Scatterv(arr[i], count, displacement, MPI_INT, bArr[i],
count[rank], MPI_INT, 0, MPI_COMM_WORLD);
}

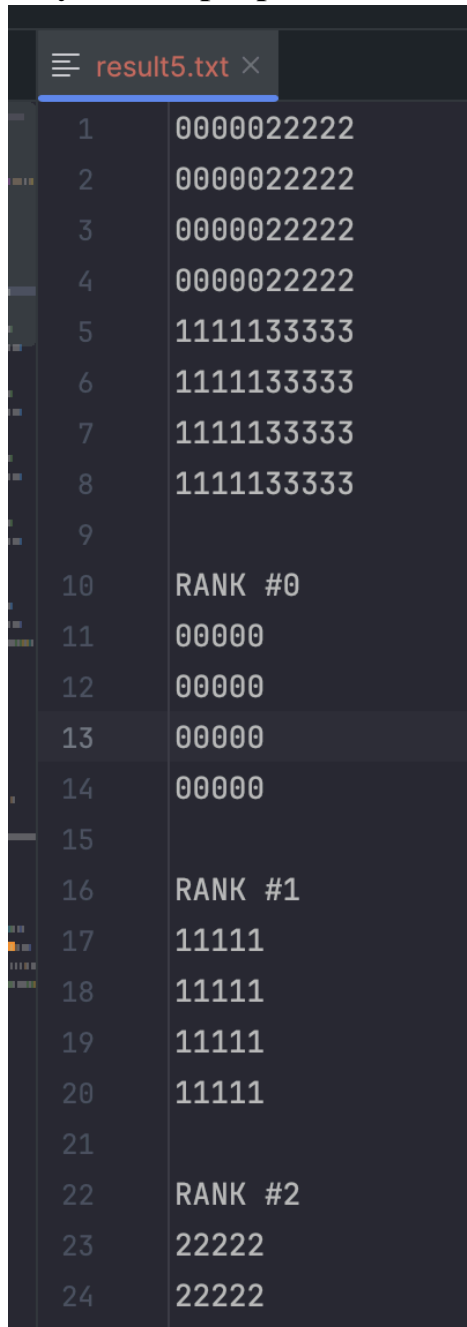
for (int i = 0; i < size; ++i)
{
    if (i == rank)
    {
        f = fopen(name, "a");
        if (f != nullptr)
        {
            fprintf(f, "RANK # %d\n", i);

            for (int i = 0; i < n / 2; ++i)
            {
                for (int j = 0; j < m / 2; ++j)
                {
                    fprintf(f, "%d", bArr[i][j]);
                }
                fprintf(f, "\n");
            }
            fprintf(f, "\n");
            fclose(f);
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
}

MPI_Finalize();
delete[] count;
delete[] displacement;
return 0;

```

### Результат программы 5:



```
1 0000022222
2 0000022222
3 0000022222
4 0000022222
5 1111133333
6 1111133333
7 1111133333
8 1111133333
9
10 RANK #0
11 00000
12 00000
13 00000
14 00000
15
16 RANK #1
17 11111
18 11111
19 11111
20 11111
21
22 RANK #2
23 22222
24 22222
```

**Вывод:** в результате работы мы познакомились с тем, как можно распределять данные между процессами с помощью функции `MPI_Scatterv`.

## Тема II. Алгоритмы параллельного программирования.

**Задание VI.** Используя алгоритм «Портфель задач», написать программу. Определить задачу, портфель, способ выхода из бесконечного цикла, решение задачи. Добавить в отчет.

### Вариант 2. Задача о голодных птенцах.

Есть  $n$  птенцов и их мать. Птенцы едят из общей миски, в которой  $F$  порций пищи. Каждый птенец съедает порцию еды, спит некоторое время, затем снова ест. Когда кончается еда, птенец, евший последним, зовет мать. Птица наполняет миску  $0 < M \leq F$  порциями еды и снова ждет, пока миска опустеет. Эти действия повторяются без конца.

Представить птиц процессами, разработать код, моделирующий их действия. Использовать алгоритм «портфель задач». Написать программы на MPI и OpenMP.

### Код программы MPI:

```
const int maxIterationsNumber = 7;
const int dishCapacity = 8;

void task1(int rank, int size, const int root = 0)
{
    srand(time(NULL));

    FILE* file;
    char fileName[30];
    int emptiesCount = 0;
    int portionCount = 6;
    int dishState = 0;
    MPI_Status status;
    const int msgtag = 0;

    if (rank == 0)
    {
        sprintf(fileName, "task1/Dish_rank_%d.txt", rank);
        file = fopen(fileName, "w");
        if (file == nullptr)
        {
            fprintf(stderr, "File error\n");
        }

        dishState = 1;
        portionCount = rand() % dishCapacity + 1;
        int bird = 2;
        int last;

        while (true)
        {
            while (portionCount > 0)
            {
                MPI_Send(&dishState, 1, MPI_INT, bird, msgtag,
MPI_COMM_WORLD);
                --portionCount;
                fprintf(file, "%d-й птенец поел из миски, еды в миске сейчас:
%d\n", -1 + bird, portionCount);
```

```

        ++bird;
        if (bird >= size)
        {
            bird = 2;
        }
    }

    fprintf(file, "Миска опустела в %d-й раз\n", emptiesCount + 1);
    dishState = -1;

    if (bird == 2)
    {
        last = size - 1;
    }
    else
    {
        last = bird - 1;
    }

    MPI_Send(&dishState, 1, MPI_INT, last, msgtag, MPI_COMM_WORLD);
    ++emptiesCount;
    MPI_Recv(&portionCount, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD,
&status);

    if (portionCount > 0)
    {
        fprintf(file, "Теперь в миске %d порций\n", portionCount);
        dishState = 1;
    }

    if (emptiesCount >= maxIterationsNumber)
    {
        dishState = 0;
        for (int i = 1; i < size; ++i)
        {
            MPI_Send(&dishState, 1, MPI_INT, i, msgtag,
MPI_COMM_WORLD);
        }

        break;
    }
}
else if (rank == 1)
{
    sprintf(fileName, "task1/Mother_rank_%d.txt", rank);
    file = fopen(fileName, "w");
    if (file == nullptr)
    {
        fprintf(stderr, "File error\n");
    }

    dishState = 0, portionCount = 0;

    while (true)
    {
        MPI_Recv(&dishState, 1, MPI_INT, MPI_ANY_SOURCE, msgtag,
MPI_COMM_WORLD, &status);
        if (dishState == -1)
        {
            fprintf(file, "Птенец № %d позвал маму в %d раз\n",
status.MPI_SOURCE - 1, emptiesCount + 1);

```

```



        portionCount = rand() % dishCapacity + 1;

        // Повар заполняет горшок
        fprintf(file, "Мама в %d-й раз заполнила миску %d порциями
еды.\n", emptiesCount++, portionCount);
        MPI_Send(&portionCount, 1, MPI_INT, 0, msgtag,
MPI_COMM_WORLD);
    }
    else if (dishState == 0)
    {
        break;
    }
}
else
{
    sprintf(fileName, "task1/Bird-%d_rank_%d.txt", rank - 1, rank);
    file = fopen(fileName, "w");
    if (file == nullptr)
    {
        fprintf(stderr, "File error\n");
    }
    dishState = 0;;
    while (true)
    {
        MPI_Recv(&dishState, 1, MPI_INT, 0, msgtag, MPI_COMM_WORLD,
&status);
        if (dishState == 1)
        {
            fprintf(file, "%d-й птенец поел из миски\n", -1 + rank);
            sleep(0.05);
        }
        else if (dishState == -1)
        {
            fprintf(file, "%d-й птенец начинает звать маму\n", -1 + rank);
            MPI_Send(&dishState, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
        }
        else
        {
            break;
        }
    }
    fclose(file);
    MPI_Finalize();
}

```

**Результат MPI:**

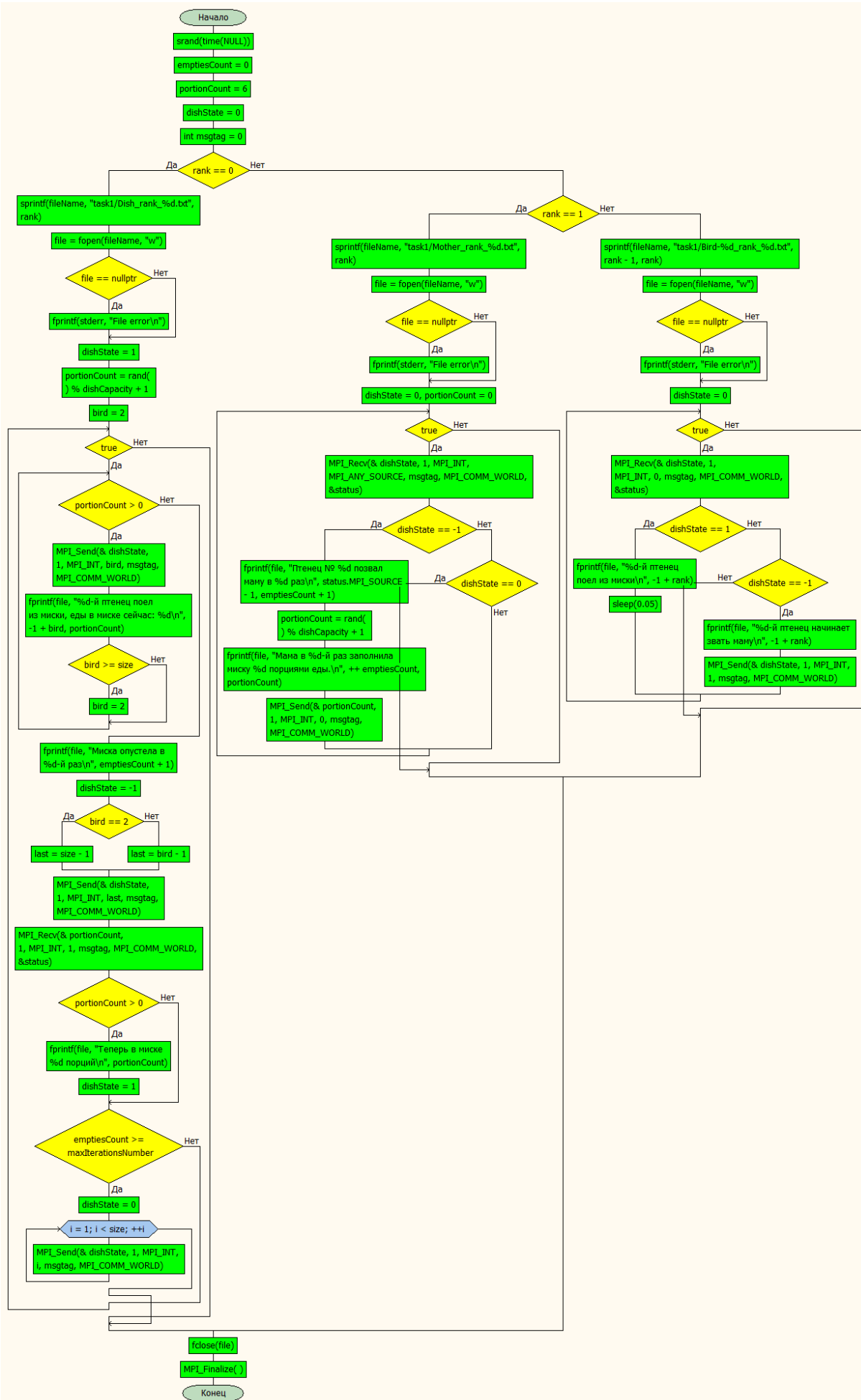
Bird-1_rank_2.txt	
1	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
2	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
3	1-й <u>птенец</u> <u>начинает</u> <u>звать</u> <u>маму</u>
4	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
5	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
6	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
7	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
8	1-й <u>птенец</u> <u>поел</u> из <u>миски</u>
9	

Dish_rank_0.txt	
1	1-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 4   16
2	2-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 3
3	3-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 2
4	4-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 1
5	1-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 0
6	<u>Миска</u> <u>опустела</u> в 1-й раз
7	<u>Теперь</u> в <u>миске</u> 5 <u>порций</u>
8	2-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 4
9	3-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 3
10	4-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 2
11	1-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 1
12	2-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 0
13	<u>Миска</u> <u>опустела</u> в 2-й раз
14	<u>Теперь</u> в <u>миске</u> 5 <u>порций</u>
15	3-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 4
16	4-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 3
17	1-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 2
18	2-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 1
19	3-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 0
20	<u>Миска</u> <u>опустела</u> в 3-й раз
21	<u>Теперь</u> в <u>миске</u> 5 <u>порций</u>
22	4-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 4
23	1-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 3
24	2-й <u>птенец</u> <u>поел</u> из <u>миски</u> , еды в <u>миске</u> <u>сейчас</u> : 2



1	<u>Птенец № 2</u> <u>позвал маму</u> в 1 раз	
2	<u>Мама</u> в 1-й раз <u>заполнила миску</u> 2 <u>порциями</u> еды.	
3	<u>Птенец № 2</u> <u>позвал маму</u> в 2 раз	
4	<u>Мама</u> в 2-й раз <u>заполнила миску</u> 1 <u>порциями</u> еды.	
5	<u>Птенец № 1</u> <u>позвал маму</u> в 3 раз	
6	<u>Мама</u> в 3-й раз <u>заполнила миску</u> 5 <u>порциями</u> еды.	
7	<u>Птенец № 2</u> <u>позвал маму</u> в 4 раз	
8	<u>Мама</u> в 4-й раз <u>заполнила миску</u> 2 <u>порциями</u> еды.	
9	<u>Птенец № 2</u> <u>позвал маму</u> в 5 раз	
10	<u>Мама</u> в 5-й раз <u>заполнила миску</u> 8 <u>порциями</u> еды.	
11	<u>Птенец № 2</u> <u>позвал маму</u> в 6 раз	
12	<u>Мама</u> в 6-й раз <u>заполнила миску</u> 3 <u>порциями</u> еды.	
13	<u>Птенец № 1</u> <u>позвал маму</u> в 7 раз	
14	<u>Мама</u> в 7-й раз <u>заполнила миску</u> 6 <u>порциями</u> еды.	

15



## Код программы OpenMP:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#include <unistd.h>

const int maxIterationsNumber = 7;
const int dishCapacity = 8;

void task2() {
    srand(time(NULL));
    int portionCount = 0;
    int emptiesCount = 0;
    int birdsCount = 5;
    int dishState = 0;
    omp_lock_t lock;

    omp_init_lock(&lock);

    FILE* file = fopen("task2/output.txt", "w");
    if (file == nullptr)
    {
        fprintf(stderr, "File error\n");
    }

#pragma omp parallel shared(portionCount, emptiesCount, dishState)
num_threads(birdsCount + 1)
    {
        int thread_id = omp_get_thread_num();

        if (thread_id == 0) {

            while (true) {
                if (emptiesCount >= maxIterationsNumber) {
                    break;
                }

                if (!dishState) {
                    omp_set_lock(&lock);
                    portionCount = rand() % dishCapacity + 1;
                    fprintf(file, "Мама в %d-й раз заполнила миску %d порциями  
еды\n", emptiesCount + 1, portionCount);
                    emptiesCount++;
                    dishState = 1;
                    omp_unset_lock(&lock);
                }
            }
        }
        else {
            while (true) {
                if (emptiesCount >= maxIterationsNumber) {
                    break;
                }

                if (dishState) {
                    omp_set_lock(&lock);
                    if (portionCount > 0) {
                        portionCount--;
                        fprintf(file, "%d-й птенец поел из миски. Порций  
осталось: %d\n", thread_id, portionCount);
                        if (portionCount <= 0) {
```

```

        if (emptiesCount < maxIterationsNumber) {
            fprintf(file, "%d-й птенец начинает звать
маму!\n", thread_id);
        }
        dishState = 0;
    }
}
omp_unset_lock(&lock);
sleep(0.5);
}
}
}
fclose(file);
omp_destroy_lock(&lock);
}

```

**Результат OpenMP:**

```
main.cpp task2.cpp output.txt x task2.h
1 Мама в 1-й раз заполнила миску 8 порциями еды
2 1-й птенец поел из миски. Порций осталось: 7
3 4-й птенец поел из миски. Порций осталось: 6
4 4-й птенец поел из миски. Порций осталось: 5
5 4-й птенец поел из миски. Порций осталось: 4
6 1-й птенец поел из миски. Порций осталось: 3
7 1-й птенец поел из миски. Порций осталось: 2
8 1-й птенец поел из миски. Порций осталось: 1
9 1-й птенец поел из миски. Порций осталось: 0
10 1-й птенец начинает звать маму!
11 Мама в 2-й раз заполнила миску 5 порциями еды
12 1-й птенец поел из миски. Порций осталось: 4
13 1-й птенец поел из миски. Порций осталось: 3
14 1-й птенец поел из миски. Порций осталось: 2
15 1-й птенец поел из миски. Порций осталось: 1
16 1-й птенец поел из миски. Порций осталось: 0
17 1-й птенец начинает звать маму!
18 Мама в 3-й раз заполнила миску 1 порциями еды
19 1-й птенец поел из миски. Порций осталось: 0
20 1-й птенец начинает звать маму!
21 Мама в 4-й раз заполнила миску 2 порциями еды
22 1-й птенец поел из миски. Порций осталось: 1
23 1-й птенец поел из миски. Порций осталось: 0
24 1-й птенец начинает звать маму!
25 Мама в 5-й раз заполнила миску 7 порциями еды
26 3-й птенец поел из миски. Порций осталось: 6
27 3-й птенец поел из миски. Порций осталось: 5
28 4-й птенец поел из миски. Порций осталось: 4
29 4-й птенец поел из миски. Порций осталось: 3
30 2-й птенец поел из миски. Порций осталось: 2
31 2-й птенец поел из миски. Порций осталось: 1
32 2-й птенец поел из миски. Порций осталось: 0
33 2-й птенец начинает звать маму!
34 Мама в 6-й раз заполнила миску 2 порциями еды
35 3-й птенец поел из миски. Порций осталось: 1
36 3-й птенец поел из миски. Порций осталось: 0
37 3-й птенец начинает звать маму!
38 Мама в 7-й раз заполнила миску 2 порциями еды
39 2-й птенец поел из миски. Порций осталось: 1
```

**Вывод:** в результате работы мы познакомились с алгоритмом «Портфель задач».

### Тема III. Технология параллельного программирования OpenMP.

**Задание VII.** Написать программу, определяющую общее количество тредов и уникальный номер каждого тредра на компьютере. Воспользоваться функциями:

`int omp_get_num_threads()` — функция возвращает общее число тредов в параллельной программе. Например, обозначим — `nthreads`;

into `omp_get_thread_num()` — функция возвращает уникальный номер треда. Это целое число в диапазоне от 0 до `nthreads - 1`.  
Добавить в отчет.

### Код программы:

```
#include "lab7.h"

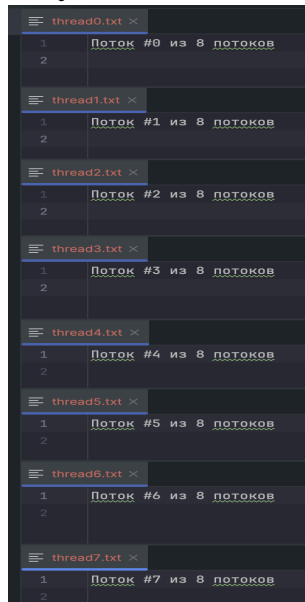
#include <omp.h>
#include <ctime>
#include <fstream>
#include <string>

void lab7()
{
    int nthreads, tid;

#pragma omp parallel private(nthreads, tid)
    {
        nthreads = omp_get_num_threads();
        tid = omp_get_thread_num();

        std::ofstream ofile("lab7/thread" + std::to_string(tid) + ".txt");
        ofile << "Поток #" << tid << " из " << nthreads << " потоков" <<
std::endl;
    }
}
```

### Результат:



**Вывод:** в результате работы мы познакомились с функциями `omp_get_num_threads` и `omp_get_thread_num`.

**Задание VIII.** Написать программу сложения матриц  $a[n][n]$  и  $b[n][n]$ , где  $n = 100$ . В результате получим матрицу  $c[n][n]$ , все элементы которой представляют собой сложенные попарно соответствующие элементы исходных матриц  $a[n][n]$  и  $b[n][n]$ . Воспользоваться директивой —

**#pragma omp parallel for schedule (dynamic)**  
**{...}**

### Код программы:

```
#include "lab8.h"

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

const int N = 100;

void writeMatrixToFile(const char* filename, int** matrix) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            fprintf(file, "%d ", matrix[i][j]);
        }
        fprintf(file, "\n");
    }

    fclose(file);
}

int** generateMatrix (int n) {
    int** matrix = new int*[n];
    for (int i = 0; i < n; i++) {
        matrix[i] = new int[n];
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matrix[i][j] = rand() % 100;
        }
    }

    return matrix;
}

void freeMatrix(int** matrix, int n) {
    for (int i = 0; i < n; i++) {
        delete[] matrix[i];
    }
    delete[] matrix;
}

void lab8() {
    int** a = generateMatrix(N);
    int** b = generateMatrix(N);
    int** c = new int*[N];
    for (int i = 0; i < N; i++) {
        c[i] = new int[N];
    }

    writeMatrixToFile("lab8/matrix-a.txt", a);
```

```

writeMatrixToFile("lab8/matrix-b.txt", b);

#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}

writeMatrixToFile("lab8/matrix-c.txt", c);

freeMatrix(a, N);
freeMatrix(b, N);
freeMatrix(c, N);
}

```

## Результат:

matrix-a.txt ×	:	matrix-b.txt ×	:	matrix-c.txt ×
1 7 49 73 58 30 0 7		1 6 68 37 9 39 66 4 0 7		1 13 117 110 67 69 138 87 94
2 1 97 71 28 37 58 77 0 7		2 20 59 22 9 90 81 41 40 7		2 21 156 93 37 127 139 118 13
3 44 95 18 96 92 15 9 0 7		3 91 54 98 89 92 23 51 75 0 7		3 135 149 116 185 184 38 142
4 53 24 30 66 0 44 70 0 7		4 73 8 58 19 35 24 77 87 0 7		4 126 32 88 85 35 68 147 172
5 49 42 49 71 10 79 8 0 7		5 82 80 18 80 6 34 86 1 8 0 7		5 131 122 67 151 16 113 169 7
6 11 16 91 71 28 71 6 0 7		6 8 79 79 72 20 37 7 85 1 0 7		6 19 95 170 143 48 108 68 130
7 47 15 93 79 29 64 7 0 7		7 25 29 52 59 84 61 0 94 0 7		7 72 44 145 138 113 125 79 95
8 52 56 70 3 39 14 99 0 7		8 66 34 94 50 29 42 69 36 0 7		8 118 90 164 53 68 56 168 96
9 3 21 75 18 0 89 62 0 7		9 60 91 78 94 60 53 13 68 0 7		9 63 112 153 112 60 142 75 12
10 78 84 42 15 60 67 4 0 7		10 83 58 99 28 20 97 93 75 0 7		10 161 142 141 43 80 164 133 8

**Вывод:** в результате работы мы воспользовались директивой **#pragma omp parallel for schedule (dynamic)**.

## Задание IX. Решение системы ОДУ на OpenMP.

Решить системы обыкновенных дифференциальных уравнений

- методом прогноз-коррекции (предиктор-корректор);

Необходимо

1. Взять последовательную программу из курса «Вычислительная математика».
2. Распараллелить программу, используя технологию OpenMP. Расчеты проводить от  $t_0=0.0$  до  $t_{max}=10.0$  с шагом  $\tau=0.001$ .
3. Сравнить результаты с последовательным вариантом.
4. Определить время вычислений.
5. Сделать вывод.
6. Добавить в отчет.



## Вариант №14.

$$\frac{dy_1}{dx} = \sin(x + y_1 y_2), \quad y_1^0 = 2,0;$$

$$\frac{dy_2}{dx} = \cos(x^2 - y_1 + y_2), \quad y_2^0 = 1,0;$$

### Код программы:

```
#include "lab9.h"
#include <iostream>
#include <cmath>
#include <omp.h>
#include <fstream>
#include <math.h>

const int N = 2;

double F1(double * y, double x) {
    return sin(x + y[0] * y[1]);
}

double F2(double* y, double x) {
    return cos(x * x - y[0] + y[1]);
}

double (*f[N])(double*, double) = { F1, F2 };

void lab9() {
    double y[N] = {2.0, 1.0};
    double yy[N] = {0.0, 0.0};
    double tmax = 10.0;
    double tau = 0.001;
    double fk[N] = {0.0}; //текущий
    double ff[N] = {0.0}; //прогнозируемый
    double t = 0.0;
    double tstart = omp_get_wtime();

#pragma omp parallel num_threads(2)
    {
        do {

#pragma omp for schedule(dynamic)
            for (int i = 0; i < N; i++) {
                fk[i] = f[i](y, t);
            }

#pragma omp for schedule(dynamic)
            for (int i = 0; i < N; i++) {
                yy[i] = y[i] + tau * fk[i];
            }
        }
    }
}
```

```

#pragma omp for schedule(dynamic)
    for (int i = 0; i < N; i++) {
        ff[i] = f[i](yy, t + tau);
    }

#pragma omp for schedule(dynamic)
    for (int i = 0; i < N; i++) {
        y[i] += tau * (fk[i] + ff[i]) / 2.0;
    }
#pragma omp master
    t += tau;
#pragma omp barrier
    } while (t <= tmax);
}

double tend = omp_get_wtime();

printf("Затраченное время на параллельное выполнение: %f\n", tend -
tstart);

for (int i = 0; i < N; i++) {
    printf("y[%d]: %f ", i, y[i]);
}
}

```

### Результат:

```

Затраченное время на последовательное выполнение: 0.000941
Затраченное время на параллельное выполнение: 0.302333
y[0]: -5.287039 y[1]: 1.005600
Process finished with exit code 0

```

**Вывод:** в результате работы мы распараллелили метод “Предиктор-корректор” и сравнили время выполнения последовательной и параллельной реализаций.

**Задание X.** Численное интегрирование методами прямоугольника в среднем и трапеций. Геометрический вид параллелизма с использованием технологии параллельного программирования OpenMP.

Выполнить численное интегрирование:

- методом прямоугольников в среднем;
- методом трапеций.

Реализовать последовательную программу и программы с использованием технологий параллельного программирования MPI и OpenMP.

Необходимо:

1. Написать последовательную программу численного интегрирования методом прямоугольников в среднем. Шаг по оси  $x$   $h = 0.00001$ . Замерить время вычислений.
2. Написать последовательную программу численного интегрирования методом трапеций. Шаг по оси  $x$   $h = 0.00001$ . Замерить время вычислений.
3. Написать параллельную программу на MPI и OpenMP численного интегрирования методом прямоугольников в среднем. Шаг по оси  $x$   $h = 0.00001$ . Замерить время вычислений.
4. Написать параллельную программу на MPI и OpenMP численного интегрирования методом трапеций. Шаг по оси  $x$   $h = 0.00001$ . Замерить время вычислений.
5. Сравнить результаты с последовательным решением.
6. Сравнить время вычислений для каждого метода.
7. Сделать вывод.
8. Добавить в отчет. Указать число использованных процессов (ядер).

#### Вариант 14. Вычислить интеграл:

$$\int_{2.0}^{3.0} \frac{1}{x \lg(x)} dx$$

#### Код программы:

```
//  
// Created by Максим Прохоров on 11.12.2024.  
// Вариант 14  
//  
  
#include "lab10.h"  
#include <iostream>  
#include <cmath>  
#include <chrono>  
#include <omp.h>  
#include <mpi.h>  
  
double func(double x) {  
    return 1/x*log10(x);  
}  
  
double rectangle_method(double a, double b, double h) {  
    double result = 0.0;  
    for (double x = a + h / 2; x < b; x += h) {  
        result += func(x) * h;  
    }  
    return result;  
}  
  
double trapezoidal_method(double a, double b, double h) {  
    double result = (func(a) + func(b)) / 2.0;  
    for (double x = a + h; x < b; x += h) {  
        result += func(x);  
    }  
}
```

```

    }
    result *= h;
    return result;
}

double rectangle_method_parallel(double a, double b, double h, int
num_threads) {
    double result = 0.0;
    int n = static_cast<int>((b - a) / h);
#pragma omp parallel for reduction(+:result) num_threads(num_threads)
    for (int i = 0; i < n; ++i) {
        double x = a + (i + 0.5) * h;
        result += func(x) * h;
    }
    return result;
}

double trapezoidal_method_parallel(double a, double b, double h, int
num_threads) {
    double result = (func(a) + func(b)) / 2.0;
    int n = static_cast<int>((b - a) / h);
#pragma omp parallel for reduction(+:result) num_threads(num_threads)
    for (int i = 1; i < n; ++i) {
        double x = a + i * h;
        result += func(x);
    }
    result *= h;
    return result;
}

double rectangle_method_mpi(double a, double b, double h, int rank, int
size) {
    double local_result = 0.0;

    double local_a = a + rank * (b - a) / size;
    double local_b = a + (rank + 1) * (b - a) / size;

    for (double x = local_a + h / 2; x < local_b; x += h) {
        local_result += func(x) * h;
    }

    return local_result;
}

double trapezoidal_method_mpi(double a, double b, double h, int rank, int
size) {
    double local_result = 0.0;

    double local_a = a + rank * (b - a) / size;
    double local_b = a + (rank + 1) * (b - a) / size;

    local_result += (func(local_a) + func(local_b)) / 2.0;
    for (double x = local_a + h; x < local_b; x += h) {
        local_result += func(x);
    }
    local_result *= h;

    return local_result;
}

void lab10(int rank, int size) {
    const double a = 2.0, b = 30.0;

```

```

const double h = 0.00001;
const int num_threads = 8;

if (rank == 0) {
    std::cout << "=== Последовательные методы ===" << std::endl;

    auto start = std::chrono::high_resolution_clock::now();
    double seq_rectangle = rectangle_method(a, b, h);
    auto end = std::chrono::high_resolution_clock::now();
    std::cout << "Метод прямоугольника: " << seq_rectangle
                << " (Время: " << std::chrono::duration<double>(end -
start).count() << "с)" << std::endl;

    start = std::chrono::high_resolution_clock::now();
    double seq_trapezoidal = trapezoidal_method(a, b, h);
    end = std::chrono::high_resolution_clock::now();
    std::cout << "Метод трапеции: " << seq_trapezoidal
                << " (Время: " << std::chrono::duration<double>(end -
start).count() << "с)" << std::endl;
}

double mpi_rectangle_result = rectangle_method_mpi(a, b, h, rank, size);
double mpi_trapezoidal_result = trapezoidal_method_mpi(a, b, h, rank,
size);

double global_rectangle_result = 0.0, global_trapezoidal_result = 0.0;

double mpi_start_time = MPI_Wtime();
mpi_rectangle_result = rectangle_method_mpi(a, b, h, rank, size);
double mpi_rectangle_time = MPI_Wtime() - mpi_start_time;

mpi_start_time = MPI_Wtime();
mpi_trapezoidal_result = trapezoidal_method_mpi(a, b, h, rank, size);
double mpi_trapezoidal_time = MPI_Wtime() - mpi_start_time;

MPI_Reduce(&mpi_rectangle_result, &global_rectangle_result, 1, MPI_DOUBLE,
MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce(&mpi_trapezoidal_result, &global_trapezoidal_result, 1,
MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (rank == 0) {
    std::cout << "\n=== Методы MPI ===" << std::endl;
    std::cout << "Метод прямоугольника (MPI): " << global_rectangle_result
<< std::endl
                << " (Время: " << mpi_rectangle_time << "с)" << std::endl;
    std::cout << "Метод трапеции (MPI): " << global_trapezoidal_result <<
std::endl
                << " (Время: " << mpi_trapezoidal_time << "с)" << std::endl;
}

if (rank == 0) {
    std::cout << "\n=== Методы OpenMP ===" << std::endl;

    double start = omp_get_wtime();
    double omp_rectangle = rectangle_method_parallel(a, b, h,
num_threads);
    double end = omp_get_wtime();
    std::cout << "Метод прямоугольника (OpenMP): " << omp_rectangle
                << " (Время: " << end - start << "с)" << std::endl;

    start = omp_get_wtime();

```

```

        double omp_trapezoidal = trapezoidal_method_parallel(a, b, h,
num_threads);
        end = omp_get_wtime();
        std::cout << "Метод трапеции (OpenMP): " << omp_trapezoidal
        << " (Время: " << end - start << "s)" << std::endl;
    }

    MPI_Finalize();
}

```

Количество ядер процессора: 8

Для наглядности использования OpenMP и MPI, нижний предел интегрирования (b) взяли за 30.

**Результат:**

```

~/ispu-ivtf-poks/3 курс/parallel_programming git:[main]
mpirun -n 4 ./cmake-build-debug/parallel_programming
=== Последовательные методы ===
Метод прямоугольника: 2.40766 (Время: 0.024448с)
Метод трапеции: 2.40766 (Время: 0.02413с)

=== Методы MPI ===
Метод прямоугольника (MPI): 2.40766 (Время: 0.007439s)
Метод трапеции (MPI): 2.40766 (Время: 0.006976s)

=== Методы OpenMP ===
Метод прямоугольника (OpenMP): 2.40766 (Время: 0.010678s)
Метод трапеции (OpenMP): 2.40766 (Время: 0.011246s)

```