Лабораторная работа №7

Передача и прием сообщений в MPI

Реализуйте процесс-«счётчик», (который запускается со значением 0 и

1) если получена -1, то он выводит в текущее значение и заканчивает работу;

2) если получено любое другое сообщение, то значение увеличивается на 1 и выводится сообщение об этом.

Код: https://pastebin.com/MG4h7qhf

```
#include <stdio.h>
#include <stdlib.h>
#include <ctime>
#include <vector>
#include "mpi.h"
using namespace std;

int getRandomNum() {
    vector<int> list{ -1, 0, 1, 2, 3 };
    int index = rand() % list.size();
    return list[index];
}

int main(int argc, char* argv[]) {
    int ProcNum, ProcRank, RecvMessage, counter;
    counter = 0;
    MPI_Status Status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    srand((unsigned)time(NULL) + ProcRank);

    int flag = 1;
    while (flag) {
        // действия, выполняемые только процессом с рангом 0
        if (ProcRank == 0) {
            for (int i = 1; i < ProcNum; i++) {
                MPI_Recv(&RecvMessage, 1, MPI_INT, i, MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
                if (RecvMessage == -1) {
                    printf("\n Value %3d recieved from %d. Stopping programm. Counter = %d", RecvMessage, i, counter);
                    flag = 0;
                }
                else {
                    if (flag > 0) {
```

```
                    counter += 1;
                    printf("\n Value %d from process %d recieved. Adding to counter. Counter = %d", RecvMessage, i, counter);
                }
            }
            MPI_Send(&flag, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    }
    // действия, выполняемые процессами, кроме процесса с рангом 0
    else {
        int message = getRandomNum();
        MPI_Send(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        MPI_Recv(&flag, 1, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
    }
}
MPI_Finalize();
return 0;
}
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 3 mpi_proj.exe

 Value 2 from process 1 recieved. Adding to counter. Counter = 1
 Value 0 from process 2 recieved. Adding to counter. Counter = 2
 Value 0 from process 1 recieved. Adding to counter. Counter = 3
 Value 3 from process 2 recieved. Adding to counter. Counter = 4
 Value  -1 recieved from 1. Stopping programm. Counter = 4
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 3 mpi_proj.exe

 Value 0 from process 1 recieved. Adding to counter. Counter = 1
 Value 3 from process 2 recieved. Adding to counter. Counter = 2
 Value 1 from process 1 recieved. Adding to counter. Counter = 3
 Value 1 from process 2 recieved. Adding to counter. Counter = 4
 Value 3 from process 1 recieved. Adding to counter. Counter = 5
 Value 2 from process 2 recieved. Adding to counter. Counter = 6
 Value 0 from process 1 recieved. Adding to counter. Counter = 7
 Value 3 from process 2 recieved. Adding to counter. Counter = 8
 Value 2 from process 1 recieved. Adding to counter. Counter = 9
 Value 1 from process 2 recieved. Adding to counter. Counter = 10
 Value 1 from process 1 recieved. Adding to counter. Counter = 11
 Value 3 from process 2 recieved. Adding to counter. Counter = 12
 Value  -1 recieved from 1. Stopping programm. Counter = 12
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 5 mpi_proj.exe

 Value 3 from process 1 recieved. Adding to counter. Counter = 1
 Value 2 from process 2 recieved. Adding to counter. Counter = 2
 Value 0 from process 3 recieved. Adding to counter. Counter = 3
 Value 3 from process 4 recieved. Adding to counter. Counter = 4
 Value  -1 recieved from 1. Stopping programm. Counter = 4
 Value  -1 recieved from 2. Stopping programm. Counter = 4
```

Контрольные вопросы.

В чем состоят основы технологии MPI?

MPI (Message Passing Interface) - стандарт для написания программ, которые выполняются на множестве процессоров. MPI предоставляет библиотеку функций для отправки и получения сообщений между процессами, работающими на разных узлах. Программы, использующие MPI, называются параллельными программами.

В чем состоят основные преимущества и недостатки технологии MPI?

Преимущества:

Высокая производительность: MPI разработан для максимальной эффективности передачи сообщений между процессорами.

Гибкость: MPI позволяет создавать программы с различными моделями параллелизма, включая разделение данных, задачи и смешанные подходы.

Переносимость: MPI работает на различных платформах и архитектурах.

Широкое распространение: MPI является стандартом де-факто для высокопроизводительных вычислений.

Недостатки:

Сложность: Программирование с использованием MPI требует определенного уровня навыков и понимания концепций параллельного программирования.

Низкий уровень абстракции: MPI предоставляет низкоуровневый API, что может быть сложным для управления и оптимизации.

Ограничения в некоторых областях: MPI не всегда является оптимальным решением для задач, требующих динамического распределения данных и задач или работы с очень большим количеством узлов.

Что понимается под параллельной программой в рамках технологии MPI?

Параллельная программа MPI состоит из нескольких процессов, выполняющихся на разных узлах. Каждый процесс имеет свой собственный адрес в памяти и может взаимодействовать с другими процессами только через передачу сообщений.

Как происходит инициализация и завершение MPI программ?

MPI_Init: Функция инициализирует среду MPI, создавая группу процессов и устанавливая коммуникатор.

MPI_Finalize: Функция завершает среду MPI, освобождая ресурсы и завершая работу всех процессов.

Как происходит передача и прием сообщений MPI программе?

MPI_Send: Функция отправляет сообщение из одного процесса в другой.

MPI_Recv: Функция принимает сообщение от другого процесса.

Лабораторная работа №8

Коллективные операции передачи данных

Модифицировать программу, написанную на Л.Р. №1, так чтобы она работала на основе коллективной передачи сообщений.

Код: https://pastebin.com/M4Jb77Af

```
#include <stdio.h>
#include <stdlib.h>
#include <ctime>
#include <vector>
#include "mpi.h"
using namespace std;

int getRandomNum() {
    vector<int> list{ -1, 0, 1, 2, 3 };
    int index = rand() % list.size();
    return list[index];
}

int main(int argc, char* argv[]) {
    int ProcNum, ProcRank;
    MPI_Status Status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    srand((unsigned)time(NULL) + ProcRank);

    // Инициализируем переменную для хранения суммарного количества сообщений
    int minMessage;
    int counter = 0;
    // Инициализируем флаг для остановки цикла
    int flag = 1;
    // Цикл, пока флаг не установлен в 0
    while (flag) {
        // Генерируем случайное число
        int message = getRandomNum();
        printf("Value %d generated in process %d.\n", message, ProcRank);
        // Выполняем операцию редукции
        MPI_Reduce(&message, &minMessage, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
        // Действия для процесса с рангом 0
        if (ProcRank == 0) {
            // Если получено сообщение с -1, останавливаем цикл
            if (minMessage == -1) {
                flag = 0;
```

```
                printf("Min value %3d recieved from all processes. Stopping programm. Iterations count = %d\n", minMessage, counter);
            }
            else {
                printf("Min value %d received from all processes. Adding to counter. Iterations count = %d\n", minMessage, counter);
                counter += 1;
            }
            // Широковещательная отправка флага всем процессам
            MPI_Bcast(&flag, 1, MPI_INT, 0, MPI_COMM_WORLD);
        }
        else {
            // Получаем флаг от процесса с рангом 0
            MPI_Bcast(&flag, 1, MPI_INT, 0, MPI_COMM_WORLD);
        }
    }
    MPI_Finalize();
    return 0;
}
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 1 mpi_proj.exe
Value 1 generated in process 0.
Min value 1 received from all processes. Adding to counter. Iterations count = 0
Value 1 generated in process 0.
Min value 1 received from all processes. Adding to counter. Iterations count = 1
Value -1 generated in process 0.
Min value  -1 recieved from all processes. Stopping programm. Iterations count = 2
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 3 mpi_proj.exe
Value -1 generated in process 1.
Value 3 generated in process 2.
Value 1 generated in process 0.
Min value  -1 recieved from all processes. Stopping programm. Iterations count = 0
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 3 mpi_proj.exe
Value 0 generated in process 1.
Value 3 generated in process 1.
Value -1 generated in process 1.
Value 2 generated in process 0.
Min value 0 received from all processes. Adding to counter. Iterations count = 0
Value 0 generated in process 0.
Min value 0 received from all processes. Adding to counter. Iterations count = 1
Value 0 generated in process 0.
Min value  -1 recieved from all processes. Stopping programm. Iterations count = 2
Value 3 generated in process 2.
Value 3 generated in process 2.
Value 0 generated in process 2.
```

Контрольные вопросы.

Как происходит передача данных от одного процесса всем?

Достижение эффективного выполнения операции передачи данных от одного процесса всем процессам программы (**широковещательная рассылка** данных) может быть обеспечено при помощи функции MPI:

int MPI_Bcast(void *buf,int count,MPI_Datatype type,int root,MPI_Comm comm);

Как происходит передача данных от всем процессов одному?

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);

Какие используются в MPI для синхронизации вычислений?

Синхронизация процессов, т.е. одновременное достижение процессами тех или иных точек процесса вычислений, обеспечивается при помощи функции MPI: int MPI_Barrier(MPI_Comm comm);

Функция MPI_Barrier должна вызываться всеми процессами используемого коммуникатора. При вызове функции MPI_Barrier выполнение процесса блокируется, продолжение вычислений процесса происходит только после вызова функции MPI_Barrier всеми процессами коммуникатора.

Как организуется неблокирующий обмен данными между процессами?

MPI обеспечивает возможность неблокированного выполнения операций передачи данных между двумя процессами. Наименование неблокирующих аналогов образуется из названий соответствующих функций путем добавления префикса I (Immediate). Список параметров неблокирующих функций содержит весь набор параметров исходных функций и один дополнительный параметр request с типом MPI_Request (в функции MPI_Irecv отсутствует также параметр status):

int MPI_Isend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Issend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Ibsend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Irsend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Irecv(void *buf, int count, MPI_Datatype type, int source, int tag, MPI_Comm comm, MPI_Request *request);

Вызов неблокирующей функции приводит к инициации запрошенной операции передачи, после чего выполнение функции завершается и процесс может продолжить свои действия. Перед своим завершением неблокирующая функция определяет переменную request, которая далее может использоваться для проверки завершения инициированной операции обмена.

Проверка состояния выполняемой неблокирующей операции передачи данных выполняется при помощи функции:

int MPI_Test(MPI_Request *request, int *flag, MPI_status *status);

Как организуется одновременное выполнение прием и передачи данных?

Достижение эффективного и гарантированного одновременного выполнения операций передачи и приема данных может быть обеспечено при помощи функции MPI:

int MPI_Sendrecv(void *sbuf,int scount,MPI_Datatype stype,int dest, int stag, void *rbuf,int rcount,MPI_Datatype rtype,int source,int rtag, MPI_Comm comm, MPI_Status *status);

Лабораторная работа №9

Обобщенная передача данных

Модифицировать программу, написанную на Л.Р. №1 так чтобы она работала на основе обобщенной передачи сообщений. Результаты работы сравнить (с результатами, полученными в Л.Р. №2) и занести в отчет.

Код: https://pastebin.com/5s7BKfJv

```
#include <stdio.h>
#include <stdlib.h>
#include <ctime>
#include <vector>
#include "mpi.h"

using namespace std;

int getRandomNum() {
    vector<int> list{ -1, 0, 1, 2, 3 };
    int index = rand() % list.size();
    return list[index];
}

int main(int argc, char* argv[]) {
    int ProcNum, ProcRank;
    MPI_Status Status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    srand((unsigned)time(NULL) + ProcRank);

    // Инициализируем переменную для хранения суммарного количества сообщений
    int minMessage;
    int counter = 0;
    // Инициализируем флаг для остановки цикла
    int flag = 1;
    // Цикл, пока флаг не установлен в 0
    while (flag) {
        // Генерируем случайное число
        int message = getRandomNum();
        printf("Value %d generated in process %d.\n", message, ProcRank);

        // Используем MPI_Gather для сбора сообщений от всех процессов
        int* allMessages = new int[ProcNum];
        MPI_Gather(&message, 1, MPI_INT, allMessages, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
        // Действия для процесса с рангом 0
        if (ProcRank == 0) {
            // Находим минимальное значение среди собранных сообщений
            minMessage = allMessages[0];
            for (int i = 1; i < ProcNum; i++) {
                if (allMessages[i] < minMessage) {
                    minMessage = allMessages[i];
                }
            }

            // Если получено сообщение с -1, останавливаем цикл
            if (minMessage == -1) {
                flag = 0;
                printf("Min value %3d received from all processes. Stopping programm. Iterations count = %d\n", minMessage, counter);
            }
            else {
                counter += 1;
                printf("Min value %d received from all processes. Adding to counter. Iterations count = %d\n", minMessage, counter);
            }
            // Широковещательная отправка флага всем процессам
            MPI_Bcast(&flag, 1, MPI_INT, 0, MPI_COMM_WORLD);

            // Освобождаем память
            delete[] allMessages;
        }
        else {
            // Получаем флаг от процесса с рангом 0
            MPI_Bcast(&flag, 1, MPI_INT, 0, MPI_COMM_WORLD);
        }
    }
    MPI_Finalize();
    return 0;
}
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 1 mpi_proj.exe
Value 3 generated in process 0.
Min value 3 received from all processes. Adding to counter. Iterations count = 1
Value 2 generated in process 0.
Min value 2 received from all processes. Adding to counter. Iterations count = 2
Value 3 generated in process 0.
Min value 3 received from all processes. Adding to counter. Iterations count = 3
Value 3 generated in process 0.
Min value 3 received from all processes. Adding to counter. Iterations count = 4
Value 1 generated in process 0.
Min value 1 received from all processes. Adding to counter. Iterations count = 5
Value -1 generated in process 0.
Min value  -1 received from all processes. Stopping programm. Iterations count = 5
```

```
D:\visual studio\проекты\mpi_proj\x64\Debug>mpiexec -n 3 mpi_proj.exe
Value 0 generated in process 1.
Value 1 generated in process 1.
Value -1 generated in process 1.
Value 3 generated in process 2.
Value 0 generated in process 2.
Value 3 generated in process 2.
Value 2 generated in process 0.
Min value 0 received from all processes. Adding to counter. Iterations count = 1
Value 3 generated in process 0.
Min value 0 received from all processes. Adding to counter. Iterations count = 2
Value 3 generated in process 0.
Min value  -1 received from all processes. Stopping programm. Iterations count = 2
```