**Лабораторная работа №2**

**Коллективные операции передачи данных**

**Цель:** изучить основные принципы коллективных операций передачи данных в технологии MPI на примере использования в рамках языка С++.

**Лабораторные задания**

**Задание.** Модифицировать программу, написанную на Л.Р. №1, так чтобы она работала на основе коллективной передачи сообщений. **Результаты работы сравнить и занести в отчет.**

#include <stdio.h>

#include "mpi.h"

#include <iostream>

#include <time.h>

#include <locale>

using namespace std;

int generate\_int(int min\_v, int max\_v) {

return min\_v + rand() % (max\_v - min\_v);

}

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

{

int ProcNum, ProcRank;

int RecvRank = 0;

MPI\_Init(&argc, &argv);

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

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

MPI\_Status Status;

bool flag = true;

int counter = 0;

int rand\_send = 0;

srand(time(0));

int global\_min;

while (flag) {

rand\_send = generate\_int(-1, 7);

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

printf("\n Random value sent to 0 process: %d", rand\_send);

MPI\_Reduce(&rand\_send, &global\_min, 1, MPI\_INT, MPI\_MIN, 0, MPI\_COMM\_WORLD);

if (ProcRank == 0) {

if (global\_min == -1) {

flag = false;

}

else {

counter++;

printf("\n The counter has been increased: %3d", counter);

}

}

MPI\_Bcast(&flag, 1, MPI\_C\_BOOL, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

**Контрольные вопросы**

1. Как происходит передача данных от одного процесса всем?
2. Как происходит передача данных от всем процессов одному?
3. Какие используются в MPI для синхронизации вычислений?
4. Как организуется неблокирующий обмен данными между процессами?
5. Как организуется одновременное выполнение прием и передачи данных?

**1. 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);

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.

**2. Как происходит передача данных от всем процессов одному?**

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.

3. 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.

**4. Как организуется неблокирующий обмен данными между процессами?**

Все рассмотренные ранее функции отправки и приема сообщений являются **блокирующими**, т.е. приостанавливающими выполнение процессов до момента завершения работы вызванных функций.

В то же время при выполнении параллельных вычислений часть сообщений может быть отправлена и принята заранее до момента реальной потребности в пересылаемых данных. В таких ситуациях желательно иметь возможность выполнения функций обмена данными без блокировки процессов для совмещения процессов передачи сообщений и вычислений.

Такой **неблокирующий способ** выполнения обменов помогает уменьшать потери эффективности параллельных вычислений из-за коммуникационных операций.

MPI обеспечивает возможность неблокированного выполнения операций передачи данных между двумя процессами.

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);

где

request - дескриптор операции, определенный при вызове неблокирующей функции,

flag - результат проверки (= true, если операция завершена),

status - результат выполнения операции обмена (только для завершенной операции).

Операция проверки является неблокирующей, т.е. процесс может проверить состояние неблокирующей операции обмена и продолжить далее свои вычисления, если по результатам проверки окажется, что операция все еще не завершена. Возможная схема совмещения вычислений и выполнения неблокирующей операции обмена может состоять в следующем:

MPI\_Isend(buf,count,type,dest,tag,comm,&request);

…

do {

…

MPI\_Test(&request,&flag,&status)

} while (!flag);

Если при выполнении неблокирующей операции окажется, что продолжение вычислений невозможно без получения передаваемых данных, то может быть использована блокирующая операция ожидания завершения операции:

int MPI\_Wait(MPI\_Request \*request, MPI\_status \*status)

**5. 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);

где

sbuf, scount, stype, dest, stag - параметры передаваемого сообщения;

rbuf, rcount, rtype, source, rtag - параметры принимаемого сообщения;

comm - коммуникатор, в рамках которого выполняется передача данных;

status – структура данных с информацией о результате выполнения операции.

Функция MPI\_Sendrecv передает сообщение, описываемое параметрами (**sbuf**, **scount**, **stype**, **dest**, **stag**), процессу с рангом **dest** и принимает сообщение в буфер, определяемый параметрами (**rbuf, rcount, rtype**, **source**, **rtag**), от процесса с рангом **source**.

В функции **MPI\_Sendrecv** для передачи и приема сообщений применяются разные буфера. В случае же, когда сообщения имеют одинаковый тип, в MPI имеется возможность использования единого буфера:

int MPI\_Sendrecv\_replace (void \*buf, int count, MPI\_Datatype type, int dest,

int stag, int source, int rtag, MPI\_Comm comm, MPI\_Status\* status)