ЛР-2

[Подзаголовок документа]

Тюльников Михаил пин-32

[Год]

Контрольные вопросы1. Как происходит передача данных от одного процесса всем?

***Ответ:*** int MPI\_Bcast(void\* buf, int count, MPI\_Datatype type, int root, MPI\_Comm comm);

Функция **MPI\_Bcast** осуществляет рассылку данных из буфера **buf**, содержащего count элементов типа **type** с процесса, имеющего номер **root**, всем процессам, входящим в коммуникатор **comm**.

* функция **MPI\_Bcast** определяет коллективную операцию и, тем самым, при выполнении необходимых рассылок данных вызов функции **MPI\_Bcast** должен быть осуществлен всеми процессами указываемого коммуникатора;
* указываемый в функции **MPI\_Bcast** буфер памяти имеет различное назначение в разныхпроцессах. Для процесса с рангом **root**, с которого осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение. Для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных.

2. Как происходит передача данных от всех процессов одному?

***Ответ:*** 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.
* функция **MPI\_Reduce** определяет коллективную операцию и, тем самым, вызов функции должен быть выполнен всеми процессами указываемого коммуникатора, все вызовы функции должны содержать одинаковые значения параметров **count**, **type**, **op**, **root**, **comm**;
* передача сообщений должна быть выполнена всеми процессами, результат операции будет получен только процессом с рангом **root**;
* выполнение операции редукции осуществляется над отдельными элементами передаваемых сообщений. Так, например, если сообщения содержат по два элемента данных и выполняется операция суммирования **MPI\_SUM**, то результат также будет состоять из двух значений, первое из которых будет содержать сумму первых элементов всех отправленных сообщений, а второе значение будет равно сумме вторых элементов сообщений соответственно

3. Какие методы используются в MPI для синхронизации вычислений?

***Ответ:*** 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?

***Ответ:*** Неблокирующий обмен происходит с помощью функций:

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);

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.

5. Как организуется одновременное выполнение прием и передачи данных?

***Ответ:*** 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)

# Пример

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include "mpi.h"

#include "Source.h"

void DataInitialization(double\* x, int N);

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

{

double x[100], TotalSum, ProcSum = 0.0;

int ProcRank, ProcNum, N = 100;

MPI\_Status Status;

// инициализация

MPI\_Init(&argc, &argv);

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

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

// подготовка данных

if (ProcRank == 0)

DataInitialization(x, N);

// рассылка данных на все процессы

MPI\_Bcast(x, N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

// вычисление частичной суммы на каждом из процессов

// на каждом процессе суммируются элементы вектора x от i1 до i2

int k = N / ProcNum;

int i1 = k \* ProcRank;

int i2 = k \* (ProcRank + 1);

if (ProcRank == ProcNum - 1)

i2 = N;

for (int i = i1; i < i2; i++)

ProcSum = ProcSum + x[i];

// сборка частичных сумм на процессе с рангом 0

if (ProcRank == 0)

{

TotalSum = ProcSum;

for (int i = 1; i < ProcNum; i++)

{

MPI\_Recv(&ProcSum, 1, MPI\_DOUBLE, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD,

&Status);

TotalSum = TotalSum + ProcSum;

}

}

else // все процессы отсылают свои частичные суммы

MPI\_Send(&ProcSum, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

// вывод результата

if (ProcRank == 0)

printf("\nTotal Sum = %10.2f", TotalSum);

MPI\_Finalize();

}

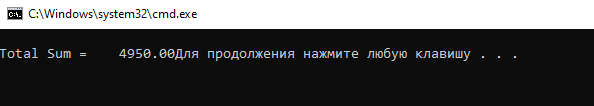
void DataInitialization(double\* x, int N) {

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

x[i] = i;

}

}



# Лабораторное задание

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "mpi.h"

#include <iostream>

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

{

int M = 4;

int ProcNum, ProcRank;

MPI\_Status Status;

MPI\_Init(&argc, &argv);

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

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

int msg = 1, recvMsg = 1;

for (int j = 0; j < M; j++) {

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

if (ProcRank == 0)

{

printf("Send %d from 0 to all process\n", msg);

}

else

{

printf("Receive %d in process %3d\n", msg, ProcRank);

msg = msg \* ProcRank;

printf("Send %d from %d to process %3d\n", msg, ProcRank, 0);

}

MPI\_Reduce(&msg, &recvMsg, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (ProcRank == 0) {

printf("Result: %d\n", recvMsg);

}

}

MPI\_Finalize();

return 0;

}

