Системы и средства параллельного программирование

Лектор: доцент Н.Н.Попова,

Тема: **«MPI. Коллективные операции передачи данных»**

11 октября 2021 г.

Тема

Организация коллективных передач в MPI

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются ВСЕМИ процессами в коммуникаторе

Классификация коллективных передач (1)

- **One-To-All**

  Один процесс определяет результат. Все процессы получают этот результат..

  - MPI_Bcast
  - MPI_Scatter, MPI_Scatterv

- **All-To-One**

  Все процессы участвуют в создании результата. Один процесс получает результат..

  - MPI_Gather, MPI_Gatherv
  - MPI_Reduce

Классификация коллективных передач (2)

- **All-To-All**

Все процессы участвуют в создании результата. Все процессы получают результат.

- MPI_Allgather, MPI_Allgatherv
- MPI_Alltoall, MPI_Alltoallv
- MPI_Allreduce, MPI_Reduce_scatter

- **Другие**

Коллективные операции, не попадающие в выше отмеченные классы.

- MPI_Scan
- MPI_Barrier

Характеристики коллективных передач

- Коллективные операции не являются помехой операциям типа «точка-точка» и наоборот
- Все процессы коммуникатора должны вызывать коллективную операцию
- Синхронизация не гарантируется (за исключением барьера).
  Завершение операции – локально в процессе
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера
- Асинхронные коллективные передачи - в MPI-3

Функции коллективных передач

Collective Communication Routines		
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce
MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv
MPI_Op_create	MPI_Op_free	MPI_Reduce
MPI_Reduce_scatter	MPI_Scan	MPI_Scatter
MPI_Scatterv		

Барьерная синхронизация

- Приостановка процессов до выхода ВСЕХ процессов коммуникатора в заданную точку синхронизации

```
int MPI_Barrier (MPI_Comm comm)
```

Пример – упорядоченный вывод:

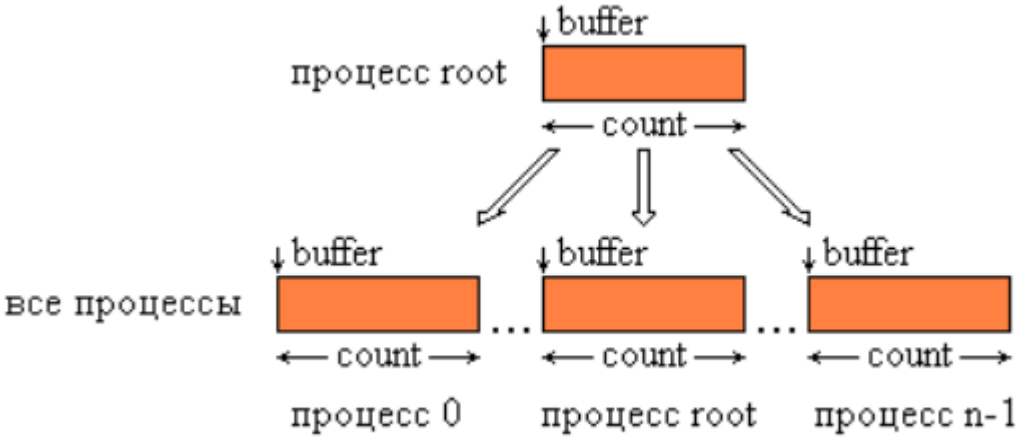
```
int size,rank;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
for (i=0;i<size;i++)
{ MPI_Barrier(MPI_COMM_WORLD);
if (rank==i) printf("%d\n", rank);
}
```

Широковещательная рассылка

- One-to-all передача: один и тот же буфер отсылается от процесса root всем остальным процессам в коммуникаторе

```
int MPI_Bcast (void *buffer, int count,
  MPI_Datatype datatype,int root, MPI_Comm comm)
```

- Все процессы должны указать один тот же root и communicator



Широковещательная рассылка

- Асинхронная широковещательная рассылка (MPI-3)

```
int MPI_Ibcast (void *buffer, int count,
    MPI_Datatype datatype,int root, MPI_Comm comm,
  MPI_Request *request)
```

Пример использования MPI_Bcast

```
#include <stdio.h>
 #include <string.h>
 #include "mpi.h“
 int main(int argc, char **argv) {
 char message[20];
int i, rank, size;
 MPI_Status status;
 int root = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 if (rank == root) { strcpy(message, "Hello, world"); }
 MPI_Bcast(message, 13, MPI_CHAR, root, MPI_COMM_WORLD);
printf( "Message from process %d : %s\n", rank, message);
 MPI_Finalize(); }
```

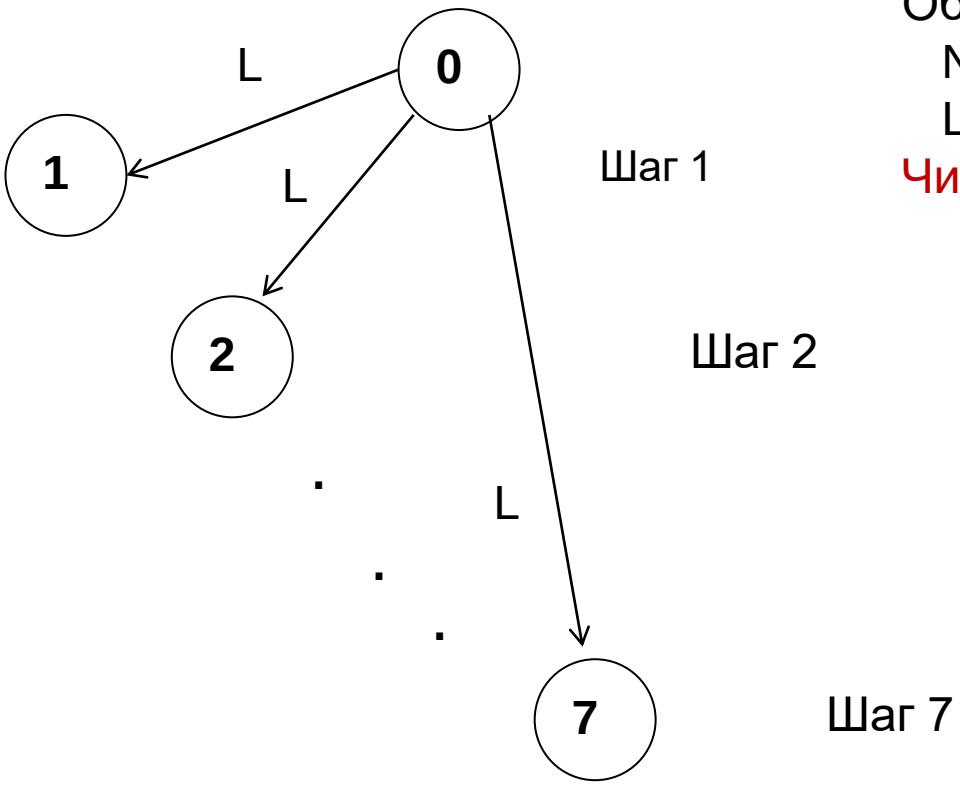
Пример использования асинхронной широковещательной рассылки (1)

```
#include <stdio.h>
 #include <string.h>
 #include "mpi.h“
 int main(int argc, char **argv) {
 char message[20];
int i, rank, size;
 MPI_Status status; MPI_Request request;
 int root = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 if (rank == root) { strcpy(message, "Hello, world"); }
 MPI_Ibcast(message, 13, MPI_CHAR, root, MPI_COMM_WORLD,
    MPI_Request *request);
printf( "Message from process %d : %s\n", rank, message);// ERROR
 MPI_Finalize(); }
```

Пример использования асинхронной широковещательной рассылки (2)

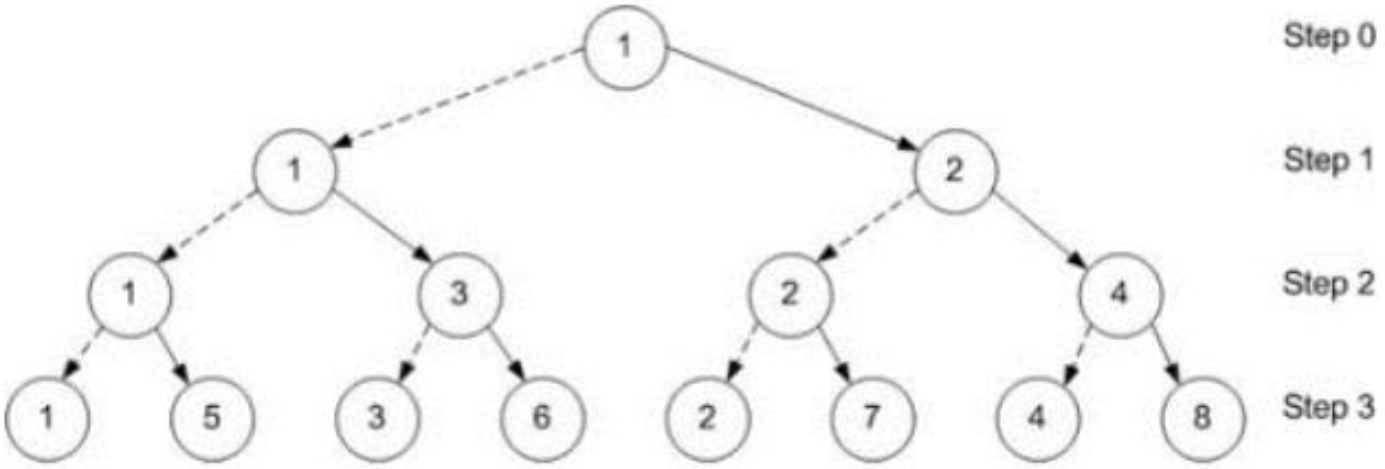
```
#include <stdio.h>
 #include <string.h>
 #include "mpi.h“
 int main(int argc, char **argv) {
 char message[20];
int i, rank, size;
 MPI_Status status; MPI_Request request;
 int root = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 if (rank == root) { strcpy(message, "Hello, world"); }
 MPI_Ibcast(message, 13, MPI_CHAR, root, MPI_COMM_WORLD,
     &request); MPI_wait (&request, &status);
printf( "Message from process %d : %s\n", rank, message);
 MPI_Finalize(); }
```

Реализация Broadcast. Один процесс рассылает всем.



Объем данных: (N-1)*L
N = число процессов
L = размер сообщения
Число шагов: N-1

Реализация Broadcast. Двоичное дерево



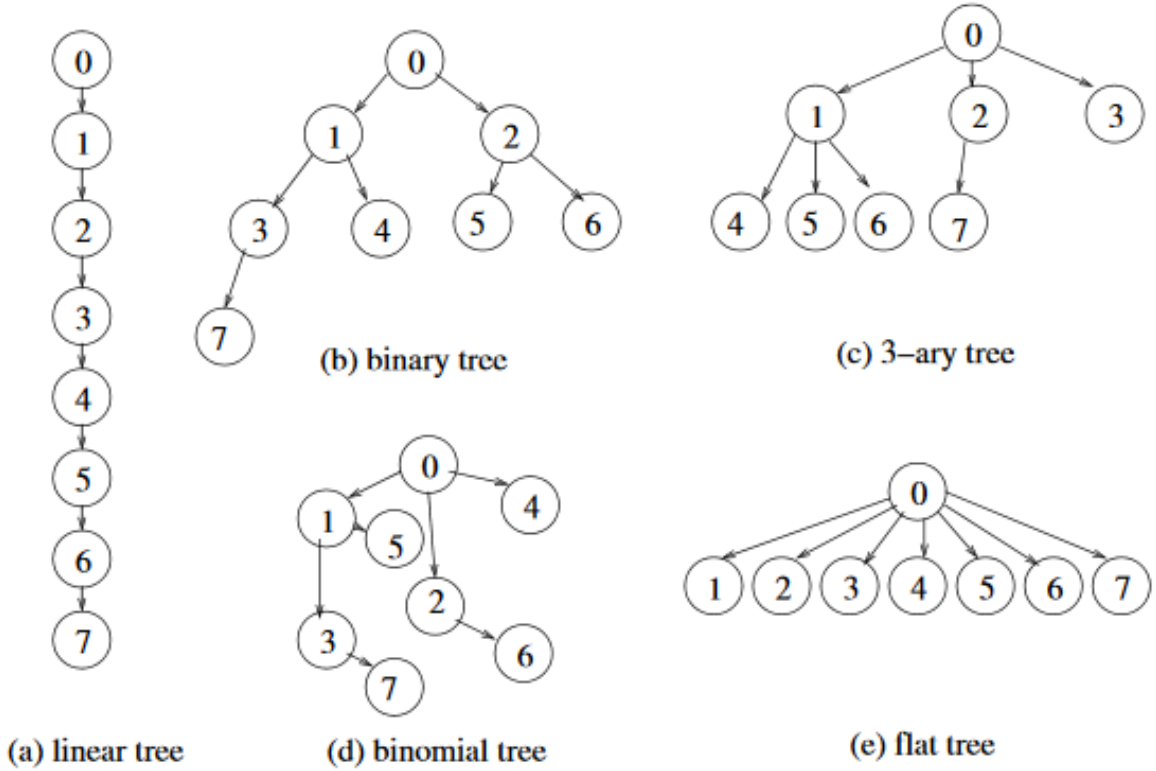
Объем данных: (N-1)*L

N = число процессов

L = размер сообщения

Число шагов: $\log_2 N$

Варианты реализации функции MPI_Broadcast



Глобальные операции редукции

- Операции выполняются над данными, распределенными по процессам коммуникатора

- Примеры:
  - Глобальная сумма или произведение
  - Глобальный максимум (минимум)
  - Глобальная операция, определенная пользователем

Общая форма

```
int MPI_Reduce(void* sendbuf, void* recvbuf,
int count, MPI_Datatype datatype, MPI_Op op,
int root,MPI_Comm comm)
```

- **count** число операций “*op*” выполняемых над последовательными элементами буфера **sendbuf**
- (также размер **recvbuf**)

- **op** является ассоциативной операцией, которая выполняется над парой операндов типа **datatype** и возвращает результат того же типа

Предопределенные операции редукции

MPI Name	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

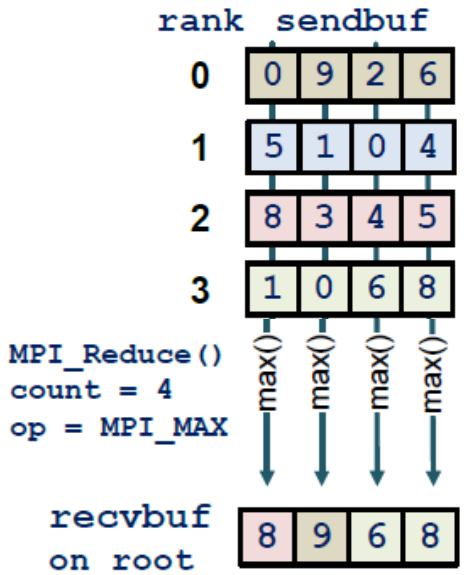
Коллективные операции редукции

Типы данных, используемых в редукционных операциях `MPI_MAXLOC` и `MPI_MINLOC`.

MPI Datatype	C Datatype
`MPI_2INT`	pair of ints
`MPI_SHORT_INT`	short and int
`MPI_LONG_INT`	long and int
`MPI_LONG_DOUBLE_INT`	long double and int
`MPI_FLOAT_INT`	float and int
`MPI_DOUBLE_INT`	double and int

Пример MPI_Reduce

MPI_Reduce (sendbuf, recvbuf, 4, MPI_INT,
MPI_MAX, 0, MPI_COMM_WORLD)



Варианты реализации MPI_Reduce

int MPI_Reduce(const void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, int root, MPI_Comm comm)

int MPI_Allreduce(const void* sbuf, void* rbuf, int count MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Reduce_scatter_block(const void* sbuf, void* rbuf, int rcount, MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Reduce_scatter(const void* sbuf, void* rbuf, const int[] rcount, MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Ireduce(const void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, int root, MPI_Comm comm, MPI_Request *request)

Варианты реализации MPI_Reduce

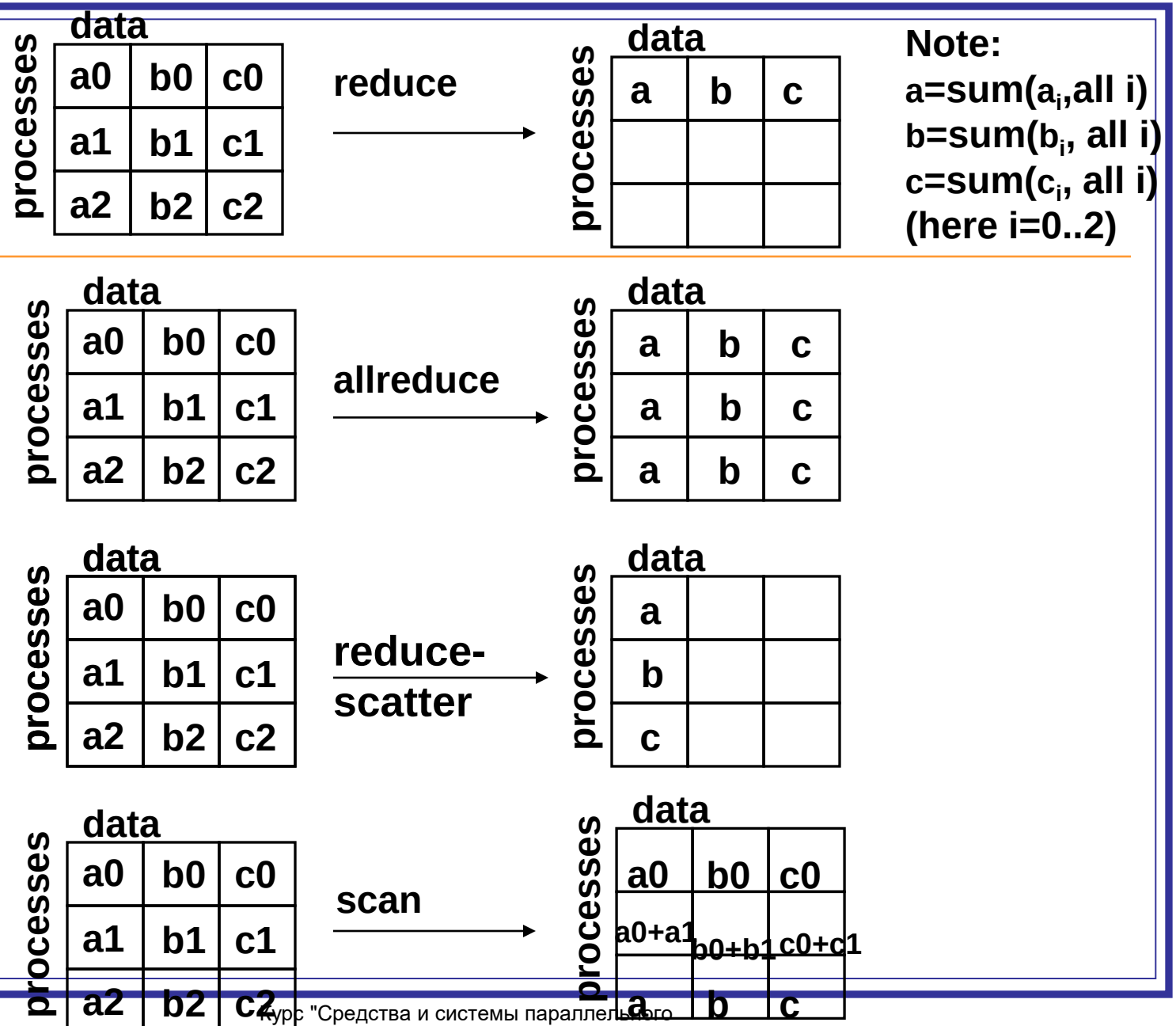
int MPI_Scan(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);

int MPI_Exscan(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);

Exclusive Scan – Scan, за исключением высывающего процесса

Детали реализации вариантов

- MPI_Reduce возвращает результат в один процесс;
- MPI_Allreduce возвращает результат всем процессам;
- MPI_Reduce_scatter_block раздает результат (вектор) по всем процессам, блоками одинакового размера.
- MPI_Reduce_scatter раздает вектор результата блоками разной длины


data
processes
a0 b0 c0
a1 b1 c1
a2 b2 c2
reduce
data
processes
a b c
Note:
a=sum(a_i,all i)
b=sum(b_i, all i)
c=sum(c_i, all i)
(here i=0..2)
data
processes
a0 b0 c0
a1 b1 c1
a2 b2 c2
allreduce
data
processes
a b c
a b c
a b c
data
processes
a0 b0 c0
a1 b1 c1
a2 b2 c2
reduce-
scatter
data
processes
a
b
c
data
processes
a0 b0 c0
a1 b1 c1
a2 b2 c2
scan
data
processes
a0 b0 c0
a0+a1 b0+b1 c0+c1
a b c

Пример использования In Place в MPI_Reduce

Неверно:

```
double result;
MPI_Reduce(&result,&result,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM
    _WORLD);
```

ОШИБКА

Верно:

```
if (rank==0)
    MPI_Reduce(&result,MPI_IN_PLACE,1,MPI_DOUBLE,MPI_SUM,0,
    MPI_COMM_WORLD);
else
    MPI_Reduce(NULL,&result,1,MPI_DOUBLE,MPI_SUM,0,
    MPI_COMM_WORLD);
```

Вычисление числа π с использованием MPI. Вариант 2.

```
#include "mpi.h"
#include <stdio.h>
int main (int argc, char *argv[])
{
    int n =100000, myid, numprocs, i;
    double mypi, pi, h, sum, x;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    h   = 1.0 / (double) n;
    sum = 0.0;
```

Вычисление числа π с использованием MPI (2)

```
for (i = myid + 1; i <= n; i += numprocs)
  {
            x = h * ((double)i - 0.5);
            sum += (4.0 / (1.0 + x*x));
   }
   mypi = h * sum;
   MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
              0, MPI_COMM_WORLD);
   if (myid == 0) printf("pi is approximately
                          %.16f", pi);
   MPI_Finalize();
   return 0;
}
```