

Параллельное программирование для высокопроизводительных вычислительных систем

Лектор: доцент Н.Н.Попова,

Лекция 8.

21 октября 2021 г.

Тема

1. Коллективные передачи в MPI (продолжение)
2. Виртуальные топологии

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются 30 MPI процессами в коммутаторе

Прошлая лекция

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 вышеотмеченные 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		
<u>MPI Allgather</u>	<u>MPI Allgatherv</u>	<u>MPI Allreduce</u>
<u>MPI Alltoall</u>	<u>MPI Alltoallv</u>	<u>MPI Barrier</u>
<u>MPI Bcast</u>	<u>MPI Gather</u>	<u>MPI Gatherv</u>
<u>MPI Op create</u>	<u>MPI Op free</u>	<u>MPI Reduce</u>
<u>MPI Reduce scatter</u>	<u>MPI Scan</u>	<u>MPI Scatter</u>
<u>MPI Scatterv</u>		

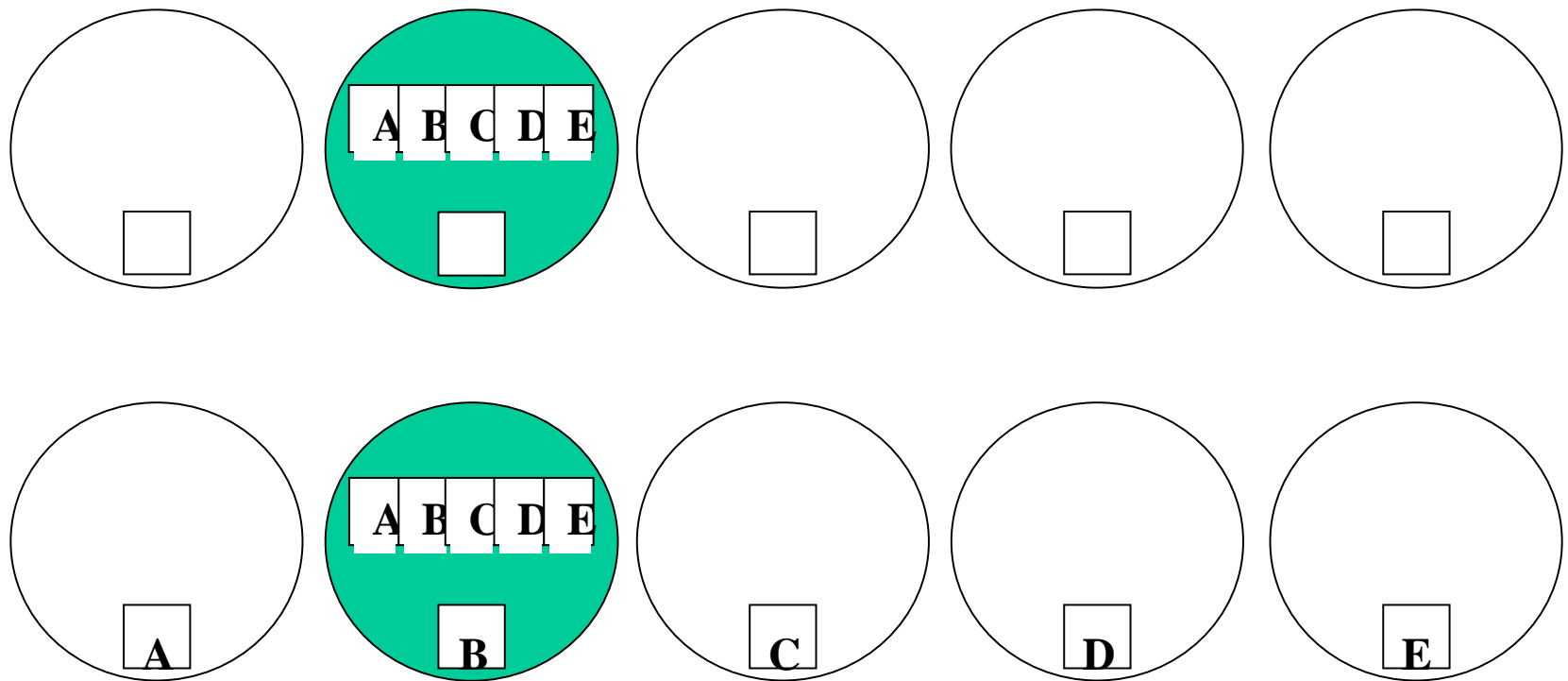
Функция Scatter рассылки блоков данных

- One-to-all передачи: блоки данных одного размера из одного процесса рассылаются всем процессам коммутатора (в порядке их номеров)

```
int MPI_Scatter(void* sendbuf, int sendcount,  
               MPI_Datatype sendtype, void* recvbuf,  
               int recvcount, MPI_Datatype recvtype, int root,  
               MPI_Comm comm)
```

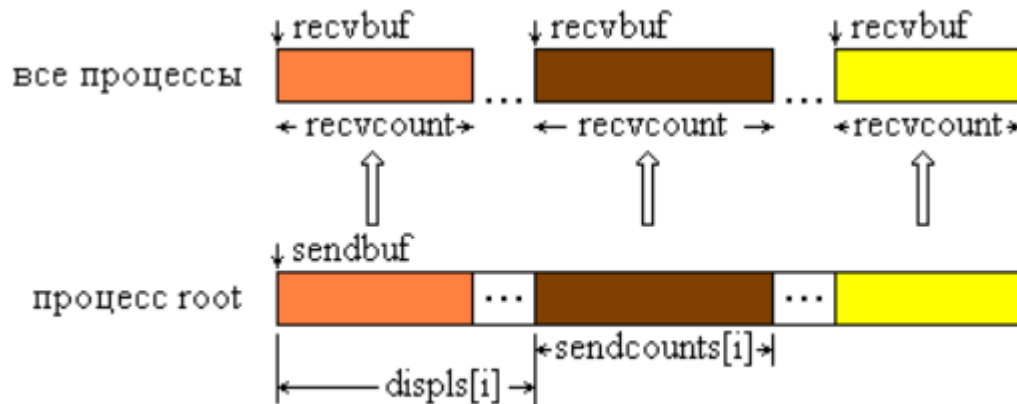
- *sendcount* – число элементов, посланных каждому процессу, **не общее число отосланных элементов**;
- send параметры (sendbuf, sendcount, sendtype) имеют смысл только для процесса root

Scatter – графическая иллюстрация

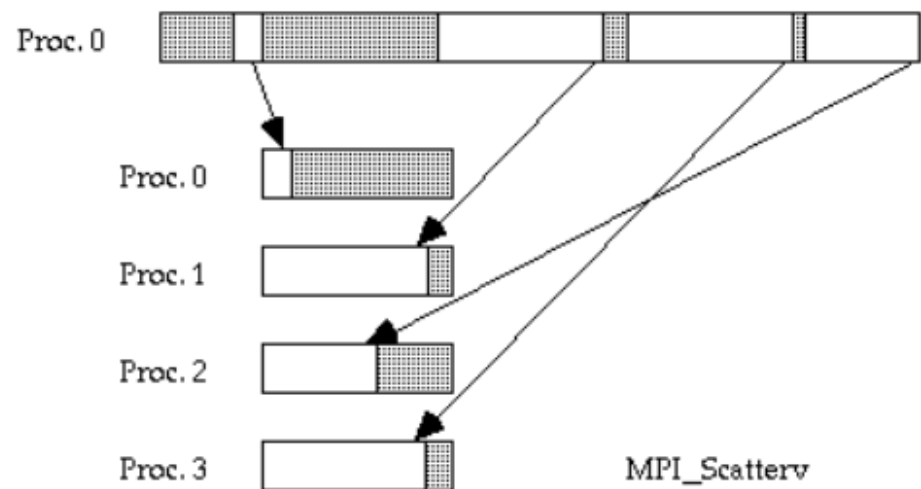
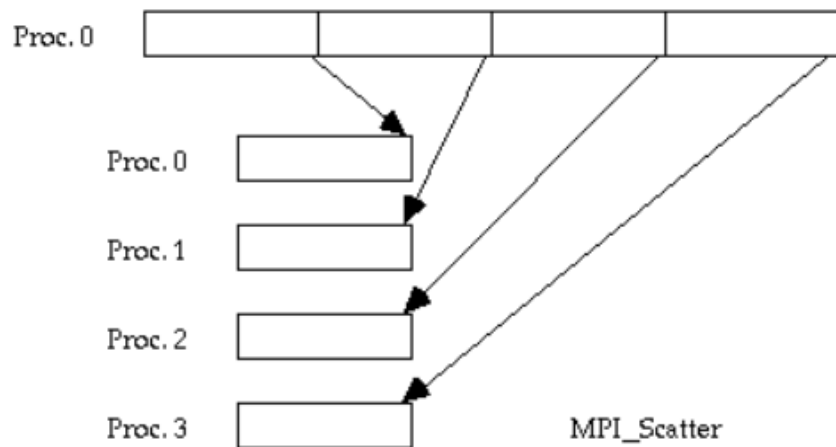


Функция Scatterv рассылки блоков разной длины

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



Сравнение: MPI_Scatter & MPI_Scatterv



Пример использования MPI_Scatterv

Рассылка массива в случае, если размер массива N НЕ ДЕЛИТСЯ нацело на число процессов size

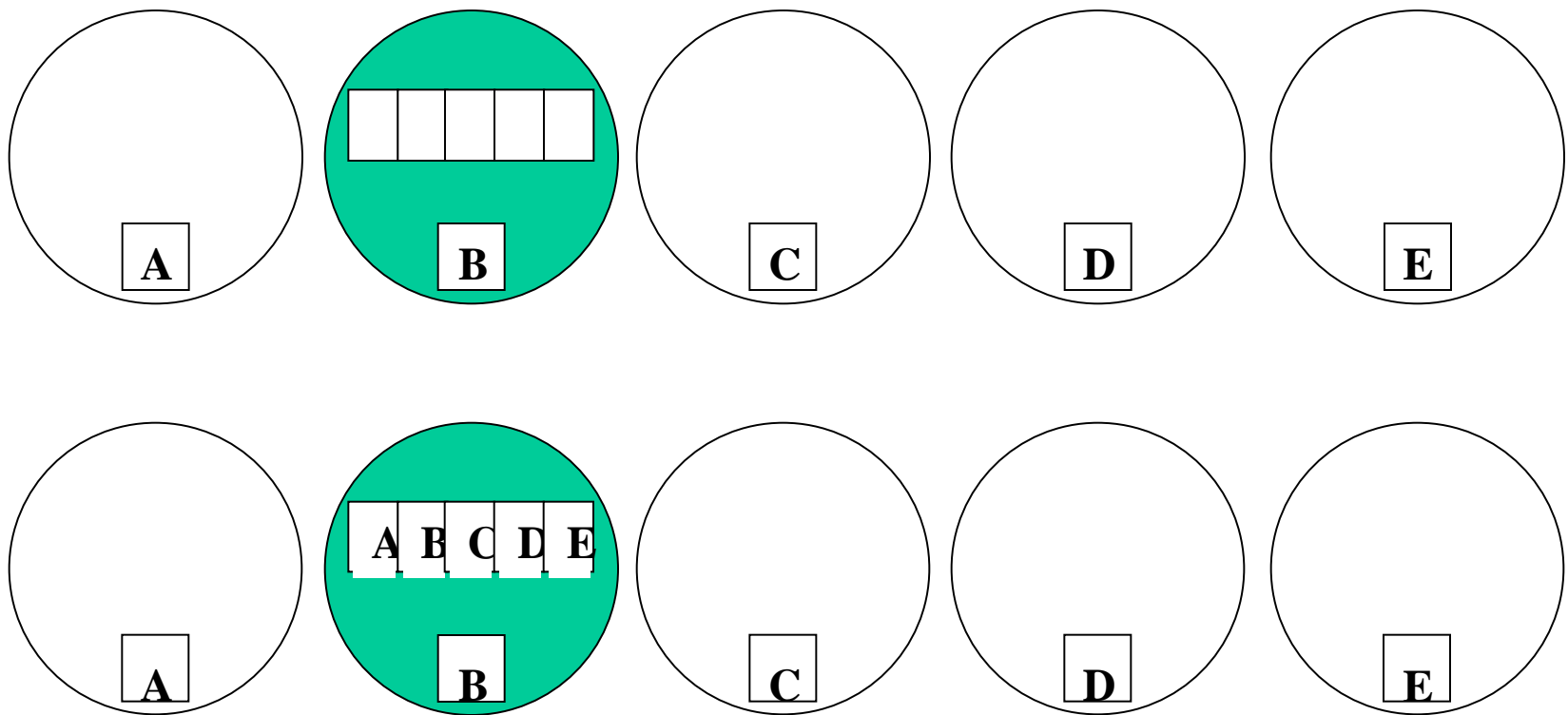
```
nmin = N/size;
nextra = N%size;
k = 0;
for (i=0; i<size; i++) {
    if (i<nextra) sendcounts[i] = nmin+1;
    else sendcounts[i] = nmin; displs[i] = k; k = k+sendcounts[i]; }
// need to set recvcnt also ...
MPI_Scatterv( sendbuf, sendcounts, displs, ...
```

Функция Gather сбора данных

- All-to-one передачи: блоки данных одинакового размера собираются процессом root
- Сбор данных выполняется в порядке номеров процессов
- Длина блоков предполагается одинаковой, т.е. данные, посланные процессом i из своего буфера `sendbuf`, помещаются в i -ю порцию буфера `recvbuf` процесса root. Длина массива, в который собираются данные, должна быть достаточной для их размещения.

```
int MPI_Gather(void* sendbuf, int sendcount,  
              MPI_Datatype sendtype,  
              void* recvbuf, int recvcount, MPI_Datatype recvtype,  
              int root, MPI_Comm comm)
```

Gather – графическая иллюстрация



Функция `Gatherv` сбора блоков данных разной длины

- Сбор данных разного размера в процессе `root` в порядке номеров процессов
- Длина блоков предполагается разной для процессов, т.е. данные, посланные процессом `i` из своего буфера `sendbuf`, помещаются в `i`-ю порцию буфера `recvbuf` процесса `root`. Начало `i`-ой порции определяется смещением, указанным в массиве `displs`. Длина массива, в который собираются данные, должна быть достаточной для их размещения.

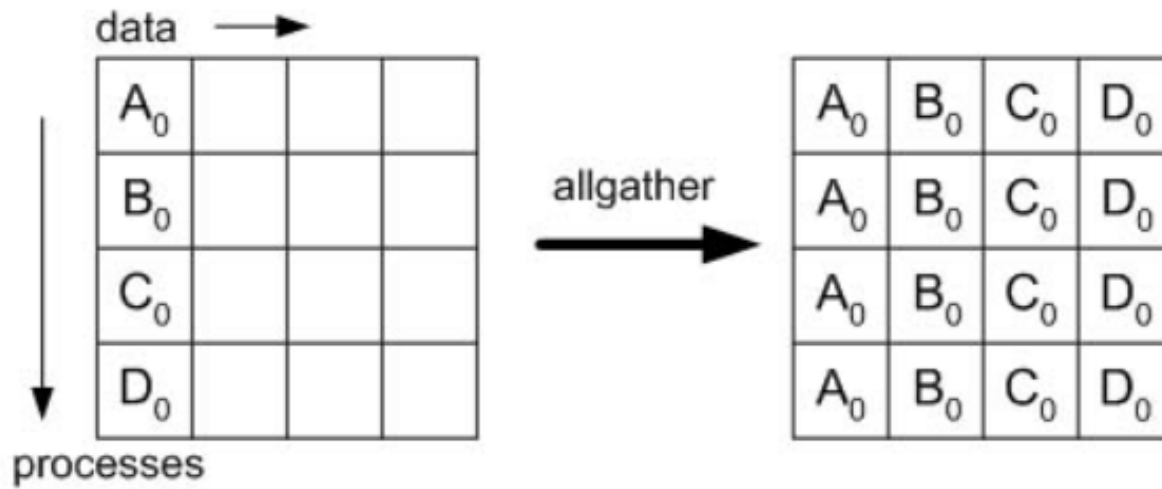
```
int MPI_Gatherv (void *sendbuf, int sendcount,  
MPI_Datatype sendtype, void *recvbuf, int *recvcounts,  
int *displs, MPI_Datatype recvtype, int root, MPI_Comm  
comm)
```

MPI_Allgather

int **MPI_Allgather**(const void* sbuf, int scount, MPI_Datatype stype, void* rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm)

int **MPI_Allgatherv**(const void* sbuf, int scounts, MPI_Datatype stype, void* rbuf, const int rcounts[], const int displs[], MPI_Datatype rtype, MPI_Comm comm)

Иллюстрация MPI_Allgather



In Place

Позволяет избежать локальное копирование, например из **send** буфер в **receive** буфер.

В качестве одного из буферов (всегда меньшего, если они различаются по размеру) можно использовать специальное значение `MPI_IN_PLACE`. Тип данных и количество передаваемых элементов этого буфера игнорируются.

Для `MPI_Gather` и `MPI_Reduce` это значение используется для **send** buffer.

Для `MPI_Scatter` это значение используется для **receive** buffer.

Пример использования In Place

```
intvalue = ...;
if (rank == root) {
    recv_buf[root] = value;
    MPI_Gather(MPI_IN_PLACE, 1, MPI_INT,
              recv_buf, 1, MPI_INT,
              root, comm);}
else {
    MPI_Gather(&value, 1, MPI_INT,
              recv_buf, 1, MPI_INT,
              root, comm);
}
```

Требуется различать root и не-root

MPI_ALLTOALL

```
int MPI_Alltoall(  
void* sendbuf, int sendcount, MPI_Datatype sendtype,  
void* recvbuf, int recvcount, MPI_Datatype recvtype,  
MPI_Comm comm);
```

Описание:

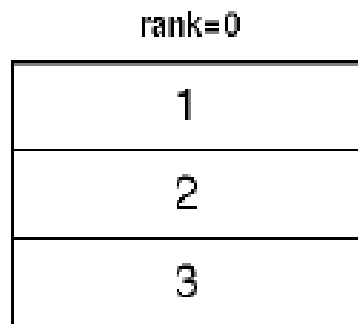
- Рассылка сообщений от каждого процесса каждому
- j-ый блок данных из процесса i принимается j-ым процессом и размещается в i-ом блоке буфера recvbuf

MPI ALLTOALL

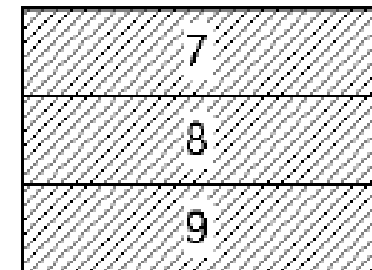
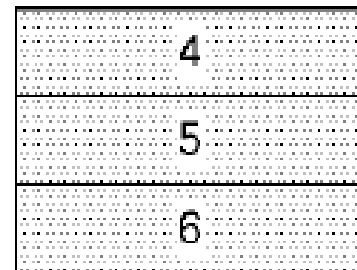
comm

sendcount

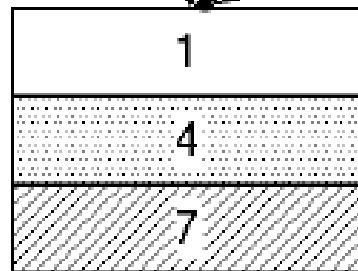
recvcount



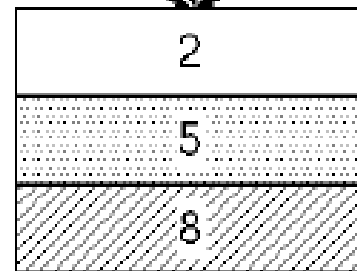
sendbuf



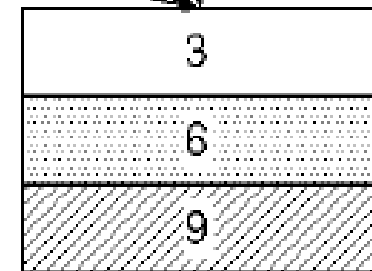
sendbuf



recvbuf



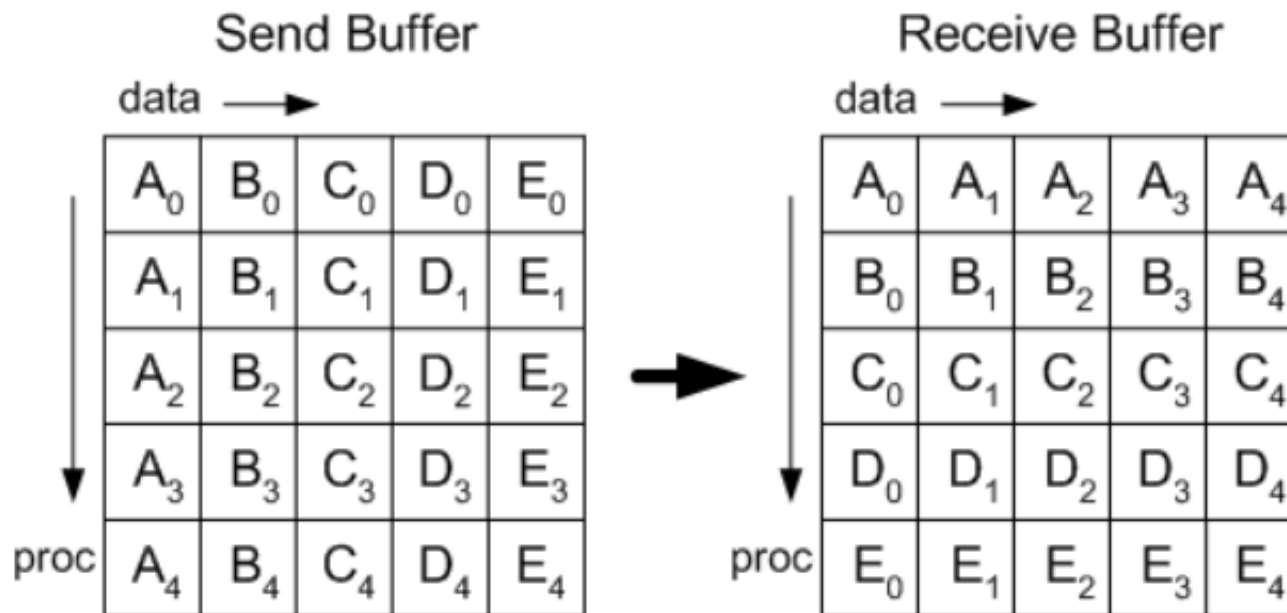
recvbuf



recvbuf

Курс "Средства и системы параллельного программирования". MPI. Коллективные функции.

MPI_ALLTOALL

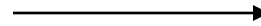


Детали реализации вариантов

- 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
- MPI_Scan префиксная редукция данных, распределенных в группе.

processes	data		
	a0	b0	c0
	a1	b1	c1
	a2	b2	c2

reduce



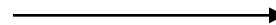
processes	data		
	a	b	c

Note:

$a = \text{sum}(a_i, \text{all } i)$
 $b = \text{sum}(b_i, \text{all } i)$
 $c = \text{sum}(c_i, \text{all } i)$
 (here $i=0..2$)

processes	data		
	a0	b0	c0
	a1	b1	c1
	a2	b2	c2

allreduce



processes	data		
	a	b	c
	a	b	c
	a	b	c

processes	data		
	a0	b0	c0
	a1	b1	c1
	a2	b2	c2

reduce-
scatter



processes	data		
	a		
	b		
	c		

processes	data		
	a0	b0	c0
	a1	b1	c1
	a2	b2	c2

scan



processes	data		
	a0	b0	c0
	a0+a1	b0+b1	c0+c1
	a	b	c

Асинхронные коллективные передачи

- Введены в MPI-3
- Семантика такая же, как и для синхронных аналогичных функций
- В список параметров добавляется параметр request :

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

```
int MPI_Ireduce(const void *sendbuf, void *recvbuf, int count,  
MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm,  
MPI_Request *request)
```

...

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);
```

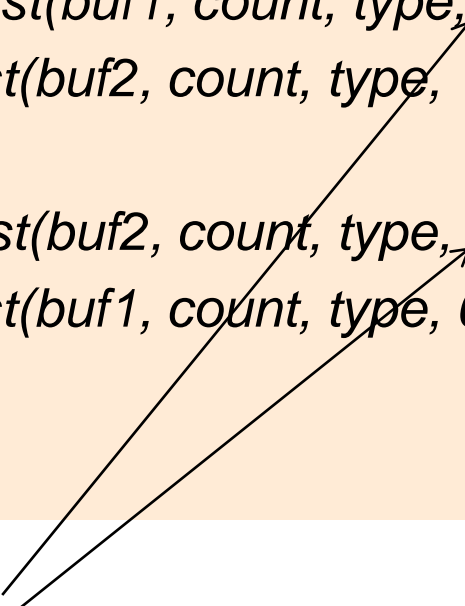
Ошибки. Пример 1.

```
switch(rank) {  
    case 0: MPI_Bcast(buf1, count, type, 0, comm);  
            MPI_Bcast(buf2, count, type, 1, comm);  
            break;  
    case 1: MPI_Bcast(buf2, count, type, 1, comm);  
            MPI_Bcast(buf1, count, type, 0, comm);  
            break;  
}
```

В чем ошибка?

Ошибки. Пример 1.

```
switch(rank) {  
    case 0: MPI_Bcast(buf1, count, type, 0, comm);  
            MPI_Bcast(buf2, count, type, 1, comm);  
            break;  
    case 1: MPI_Bcast(buf2, count, type, 1, comm);  
            MPI_Bcast(buf1, count, type, 0, comm);  
            break;  
}
```



Коллективные операции должны выполняться в одном и том же порядке всеми участниками коммуникационной группы. .

Ошибки. Пример 2.

```
switch(rank) {  
  case 0: MPI_Bcast(buf1, count, type, 0, comm);  
          MPI_Send(buf2, count, type, 1, tag, comm);  
          break;  
  case 1: MPI_Recv(buf2, count, type, 0, tag, comm, status);  
          MPI_Bcast(buf1, count, type, 0, comm);  
          break;  
}
```

В чем ошибка?

Ошибки. Пример 2.

```
switch(rank) {  
  case 0: MPI_Bcast(buf1, count, type, 0, comm);  
          MPI_Send(buf2, count, type, 1, tag, comm);  
          break;  
  case 1: MPI_Recv(buf2, count, type, 0, tag, comm, status);  
          MPI_Bcast(buf1, count, type, 0, comm);  
          break;  
}
```

Относительный порядок выполнения коллективных операций и операций «точка-точка» должен быть таким, чтобы даже в случае синхронизации коллективных операций и операций «точка-точка» не возникало тупиковой ситуации.

Тема

- Виртуальные топологии

Понятие коммуникатора MPI

- Коммуникатор - управляющий объект, представляющий группу процессов, которые могут взаимодействовать друг с другом

Группы и коммутаторы

■ Группа:

- Упорядоченное множество процессов
- Каждый процесс в группе имеет уникальный номер
- Процесс может принадлежать нескольким группам
 - rank всегда относителен группы

■ Коммутаторы:

- Все обмены сообщений всегда проходят в рамках коммутатора
 - С точки зрения программирования группы и коммутаторы эквивалентны
 - Коммутаторы – глобальные объекты.
- Группы и коммутаторы – динамические объекты, должны создаваться и уничтожаться в процессе работы программы

Специальные типы MPI

- MPI_Comm
- MPI_group

Создание новых коммуникаторов

2 способа создания новых коммуникаторов:

- Использовать функции для работы с группами и коммуникаторами (создать новую группу процессов и по новой группе создать коммуникатор, разделить коммуникатор и т.п.)
- Использовать встроенные в MPI виртуальные топологии

Предопределенные коммутаторы

- MPI-1 поддерживает 3 предопределенных коммутатора:
 - MPI_COMM_WORLD
 - MPI_COMM_NULL
 - MPI_COMM_SELF
- Только MPI_COMM_WORLD используется для передачи сообщений
- Нужны для реализации ряда функций MPI

Использование MPI_COMM_WORLD

- Содержит все доступные на момент старта программы процессы
- Обеспечивает начальное коммуникационное пространство
- Простые программы часто используют только MPI_COMM_WORLD
- Сложные программы дублируют и производят действия с копиями MPI_COMM_WORLD

Использование MPI_COMM_NULL

- Нет реализации такого коммуникатора
- Не может быть использован как параметр ни в одной из функций
- Может использоваться как начальное значение коммуникатора
- Возвращается в качестве результата в некоторых функциях
- Возвращается как значение после операции освобождения коммуникатора

Использование MPI_COMM_SELF

- Содержит только локальный процесс
- Обычно не используется для передачи сообщений
- Содержит информацию:
 - кэшированные атрибуты, соответствующие процессу
 - предоставление единственного входа для определенных ВЫЗОВОВ

Виртуальные топологии

- Удобный способ именования процессов
- Упрощение написания параллельных программ
- Оптимизация передач
- Возможность выбора топологии, соответствующей логической структуре задачи

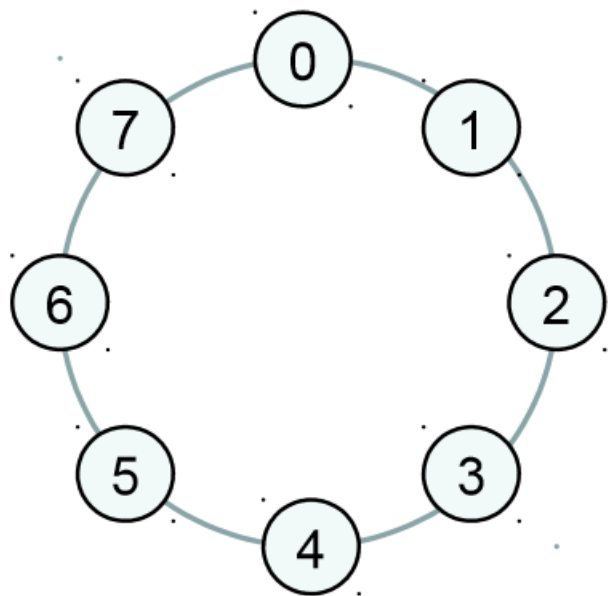
Как использовать виртуальные ТОПОЛОГИИ

- Создание топологии – новый коммуникатор
- MPI обеспечивает “mapping functions”
- Mapping функции вычисляют ранг процессов, базирясь на топологии

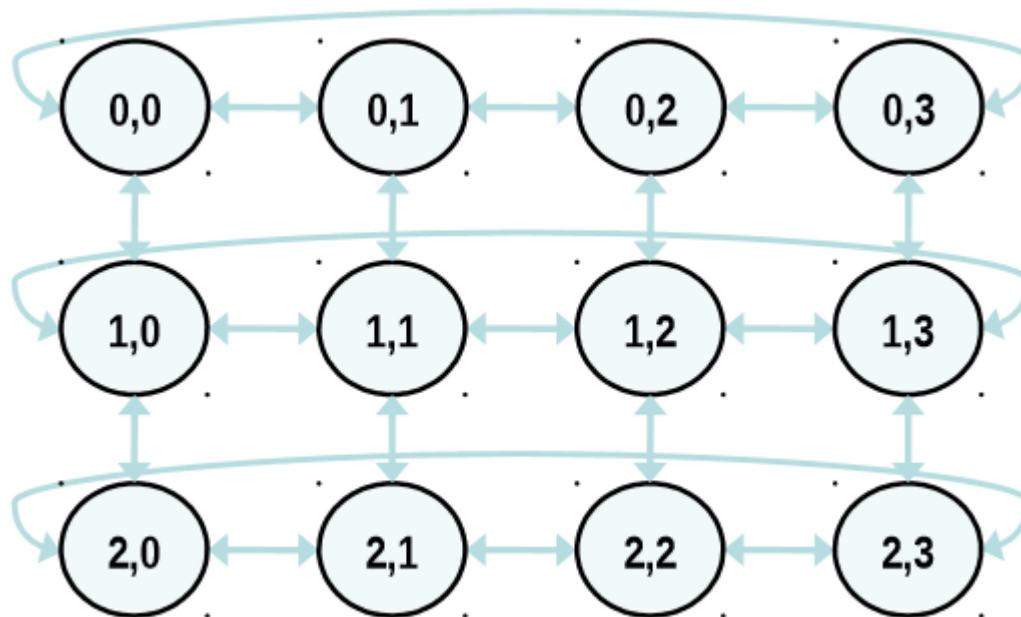
Типы виртуальных топологий

- Декартовская (многомерная решетка)
- Графовая

Пример декартовых топологий



1D



2D

Отображение данных на виртуальную топологию

