Основы параллельного программирования с использованием MPI

Лекция 5

Немнюгин Сергей Андреевич

Санкт-Петербургский государственный университет

физический факультет

кафедра вычислительной физики

Интернет-Университет
Суперкомпьютерных Технологий

High-Performance Computing University

Лекция 5

Аннотация

В лекции рассматриваются коллективные обмены. Среди них – широковещательная рассылка. Обсуждаются операции распределения и сбора данных, а также операции приведения (редукции). Внимание уделяется также роли синхронизации в параллельном программировании и средствам синхронизации в MPI.

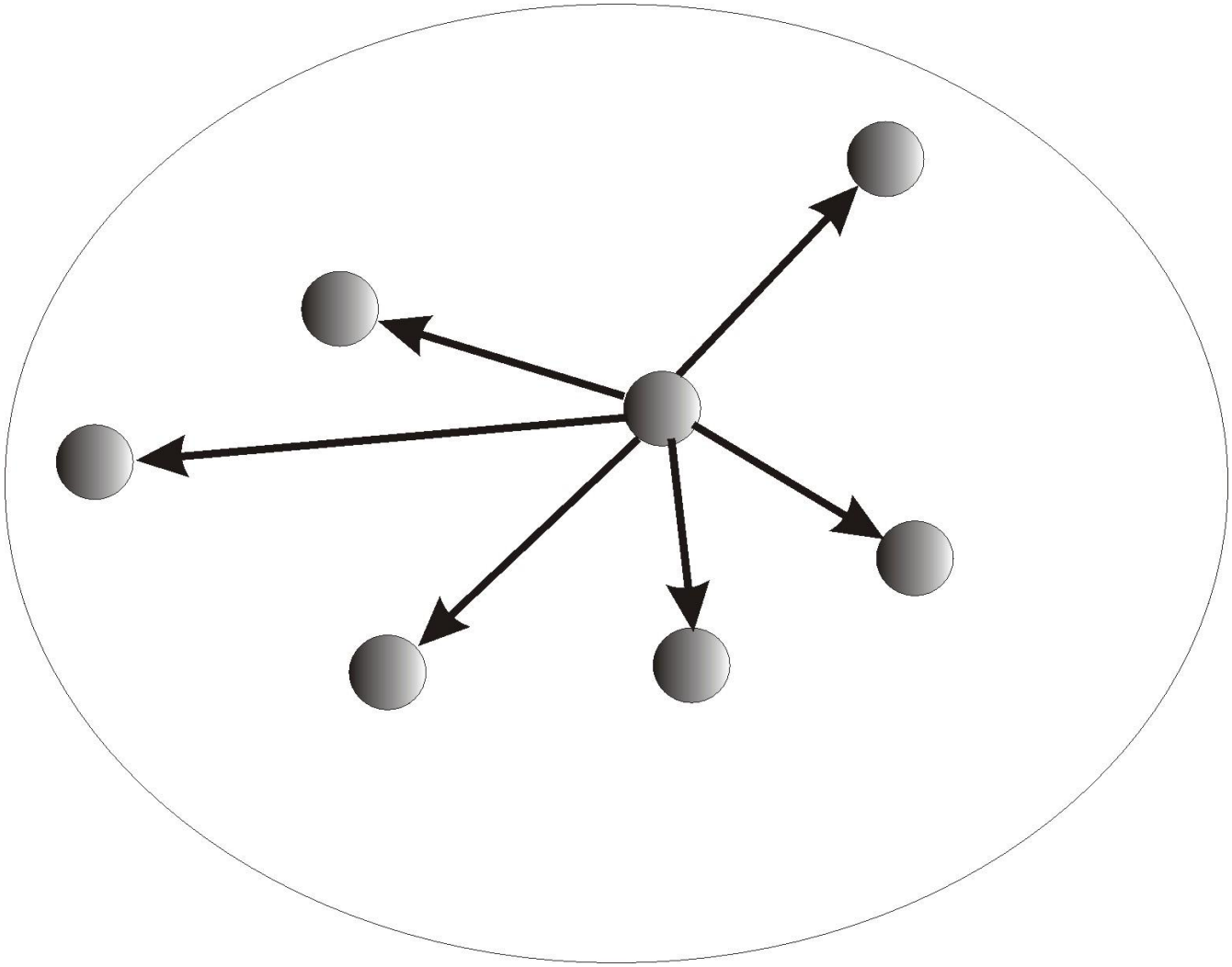
План лекции

- Особенности коллективных обменов.
- Широковещательная рассылка.
- Операции распределения и сбора данных.
- Операции приведения.
- Синхронизация.

Коллективные обмены

Коллективные обмены

В операцию коллективного обмена вовлечены не два, а большее число процессов.



Коллективные обмены

Общая характеристика коллективных обменов:

- коллективные обмены не могут взаимодействовать с двухточечными. Коллективная передача не может быть перехвачена двухточечной подпрограммой приема;
- коллективные обмены могут выполняться как с синхронизацией, так и без нее;
- все коллективные обмены являются блокирующими для инициировавшего их обмена;
- теги сообщений в коллективных обменах назначаются системой.

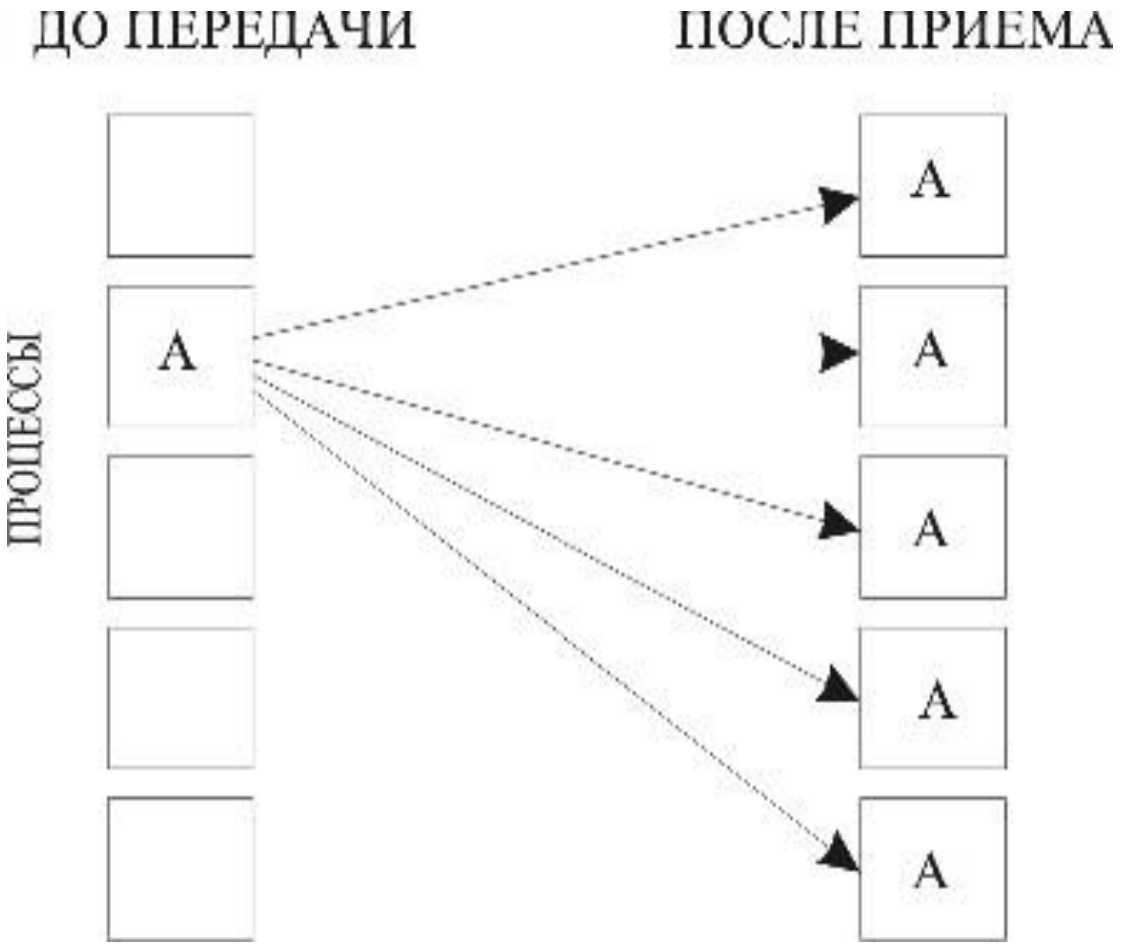
Коллективные обмены

Виды коллективных обменов:

- *широковещательная передача* - выполняется от одного процесса ко всем;
- *распределение* данных;
- *сбор* данных;
- *синхронизация с барьером* - это форма синхронизации работы процессов, когда выполнение программы продолжается только после того, как к соответствующей процедуре обратилось определенное число процессов;
- *операции приведения* - входными являются данные нескольких процессов, а результат одно значение, которое становится доступным всем процессам, участвующим в обмене;
- *операции сканирования* – операции частичного приведения.

Коллективные обмены

Широковещательная рассылка



Коллективные обмены

Широковещательная рассылка выполняется подпрограммой:

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

MPI_Bcast(buffer, count, datatype, root, comm, ierr)
```

Параметры этой процедуры одновременно являются входными и выходными:

- `buffer` - адрес буфера;
- `count` - количество элементов данных в сообщении;
- `datatype` - тип данных MPI;
- `root` - ранг главного процесса, выполняющего широковещательную рассылку;
- `comm` - коммуникатор.

Коллективные обмены

Пример 1 использования широковещательной рассылки

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 char data[24];
 int myrank, count = 25;
 MPI_Status status;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
{
 strcpy(data, "Hi, Parallel Programmer!");
 MPI_Bcast(&data, count, MPI_BYTE, 0, MPI_COMM_WORLD);
 printf("send: %s\n", data);
}
else
```

Коллективные обмены

```
 MPI_Bcast(&data, count, MPI_BYTE, 0, MPI_COMM_WORLD);
 printf("received: %s\n", data);
}
 MPI_Finalize();
 return 0;
}
```

Коллективные обмены

Пример 2 использования широковещательной пересылки

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
 int myrank;
 int root = 0;
 int count = 1;
 float a, b;
 int n;

 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
```

Коллективные обмены

```
printf("Enter a, b, n\n");
scanf("%f %f %i", &a, &b, &n);
MPI_Bcast(&a, count, MPI_FLOAT, root, MPI_COMM_WORLD);
MPI_Bcast(&b, count, MPI_FLOAT, root, MPI_COMM_WORLD);
MPI_Bcast(&n, count, MPI_INT, root, MPI_COMM_WORLD);
}
else
{
MPI_Bcast(&a, count, MPI_FLOAT, root, MPI_COMM_WORLD);
MPI_Bcast(&b, count, MPI_FLOAT, root, MPI_COMM_WORLD);
MPI_Bcast(&n, count, MPI_INT, root, MPI_COMM_WORLD);
printf("%i Process got %f %f %i\n", myrank, a, b, n);
}
 MPI_Finalize();
 return 0;
}
```

Коллективные обмены

Распределение данных

```
int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype,
int root, MPI_Comm comm)

MPI_Scatter(sendbuf, sendcount, sendtype, rcvbuf, rcvcount,
rcvtype, root, comm, ierr)
```

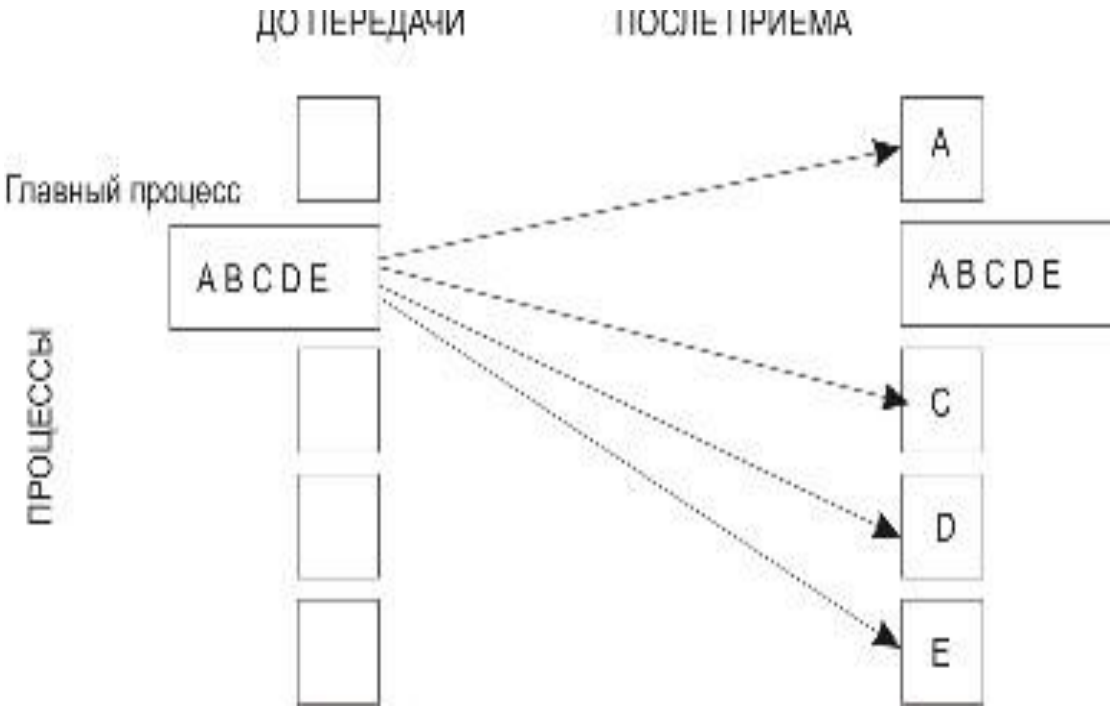
Входные параметры:

- `sendbuf` - адрес буфера передачи;
- `sendcount` - количество элементов, пересылаемых каждому процессу (не суммарное количество пересылаемых элементов!);
- `sendtype` - тип передаваемых данных;
- `rcvcount` - количество элементов в буфере приема;
- `rcvtype` - тип принимаемых данных;
- `root` - ранг передающего процесса;
- `comm` - коммуникатор.

Выходной параметр: `rcvbuf` - адрес буфера приема.

Коллективные обмены

Процесс с рангом `root` распределяет содержимое буфера передачи `sendbuf` среди всех процессов. Содержимое буфера передачи разбивается на несколько фрагментов, каждый из которых содержит `sendcount` элементов. Первый фрагмент передается процессу 0, второй процессу 1 и т. д. Аргументы `send` имеют значение только на стороне распределяющего процесса root.



Коллективные обмены

Сбор данных

Сбор данных от остальных процессов в буфер главной задачи выполняется подпрограммой:

```
int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype,
int root, MPI_Comm comm)

MPI_Gather(sendbuf, sendcount, sendtype, rcvbuf, rcvcount,
rcvtype, root, comm, ierr)
```

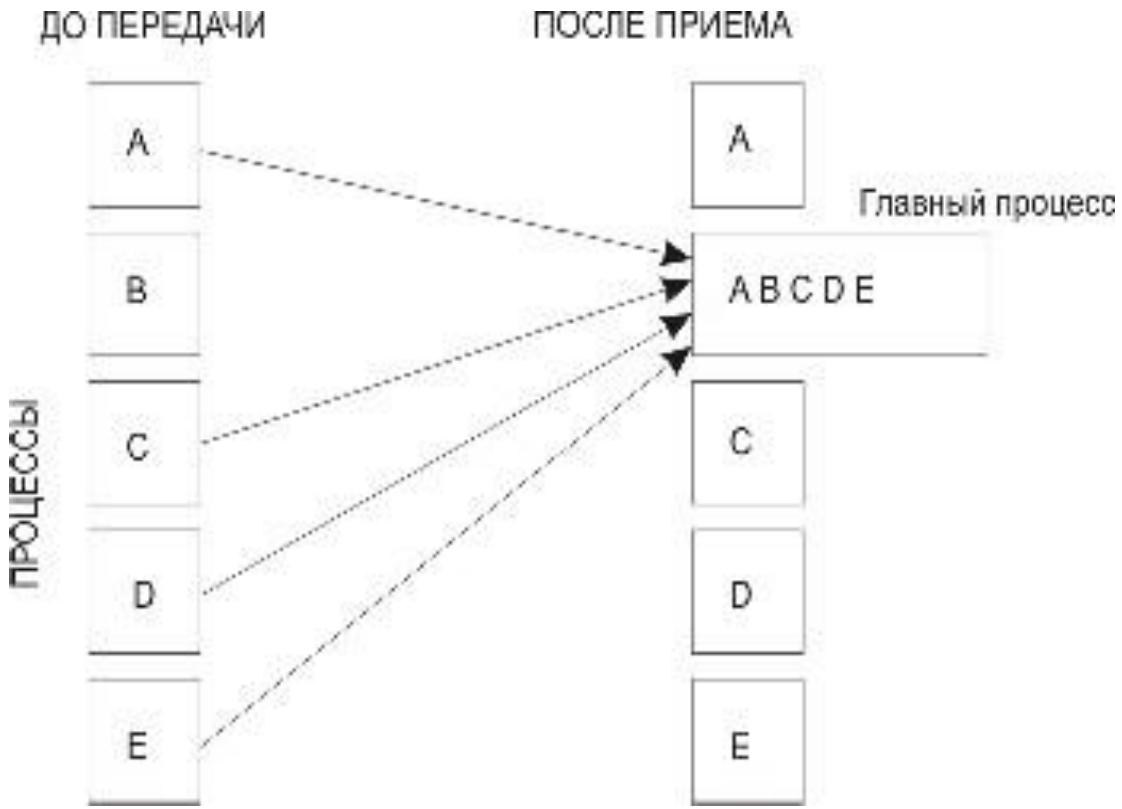
Каждый процесс в коммуникаторе comm пересылает содержимое буфера передачи `sendbuf` процессу с рангом `root`. Процесс root «склеивает» полученные данные в буфере приема.

Коллективные обмены

Порядок склейки определяется рангами процессов, то есть в результирующем наборе после данных от процесса 0 следуют данные от процесса 1, затем данные от процесса 2 и т. д. Аргументы `rcvbuf`, `rcvcount` и `rcvtype` играют роль только на стороне главного процесса. Аргумент `rcvcount` указывает количество элементов данных, полученных от каждого процесса (но не суммарное их количество). При вызове подпрограмм `MPI_Scatter` и `MPI_Gather` из разных процессов следует использовать общий главный процесс.

Коллективные обмены

Сбор данных



Коллективные обмены

Сбор данных от всех процессов и распределение их всем процессам:

```
int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype,
MPI_Comm comm)

MPI_Allgather(sendbuf, sendcount, sendtype, rcvbuf, rcvcount,
rcvtype, comm, ierr)
```

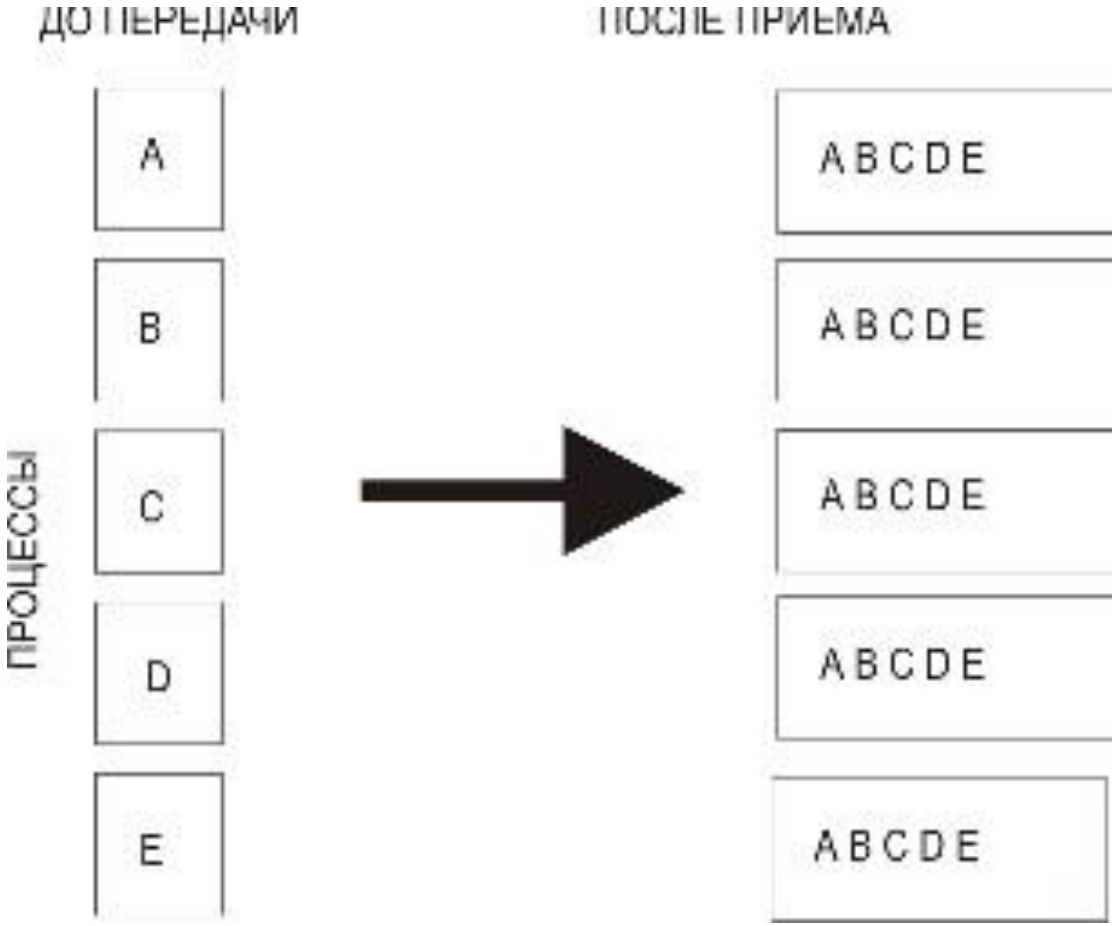
Входные параметры:

- `sendbuf` - начальный адрес буфера передачи;
- `sendcount` - количество элементов в буфере передачи;
- `sendtype` - тип передаваемых данных;
- `rcvcount` - количество элементов, полученных от каждого процесса;
- `rcvtype` - тип данных в буфере приема;
- `comm` - коммуникатор.

Выходной параметр: `rcvbuf` - адрес буфера приема.

Коллективные обмены

Блок данных, переданный от *j*-го процесса, принимается каждым процессом и размещается в *j*-м блоке буфера приема `recvbuf`.



Коллективные обмены

Операция приведения

Операция *приведения*, результат которой передается одному процессу

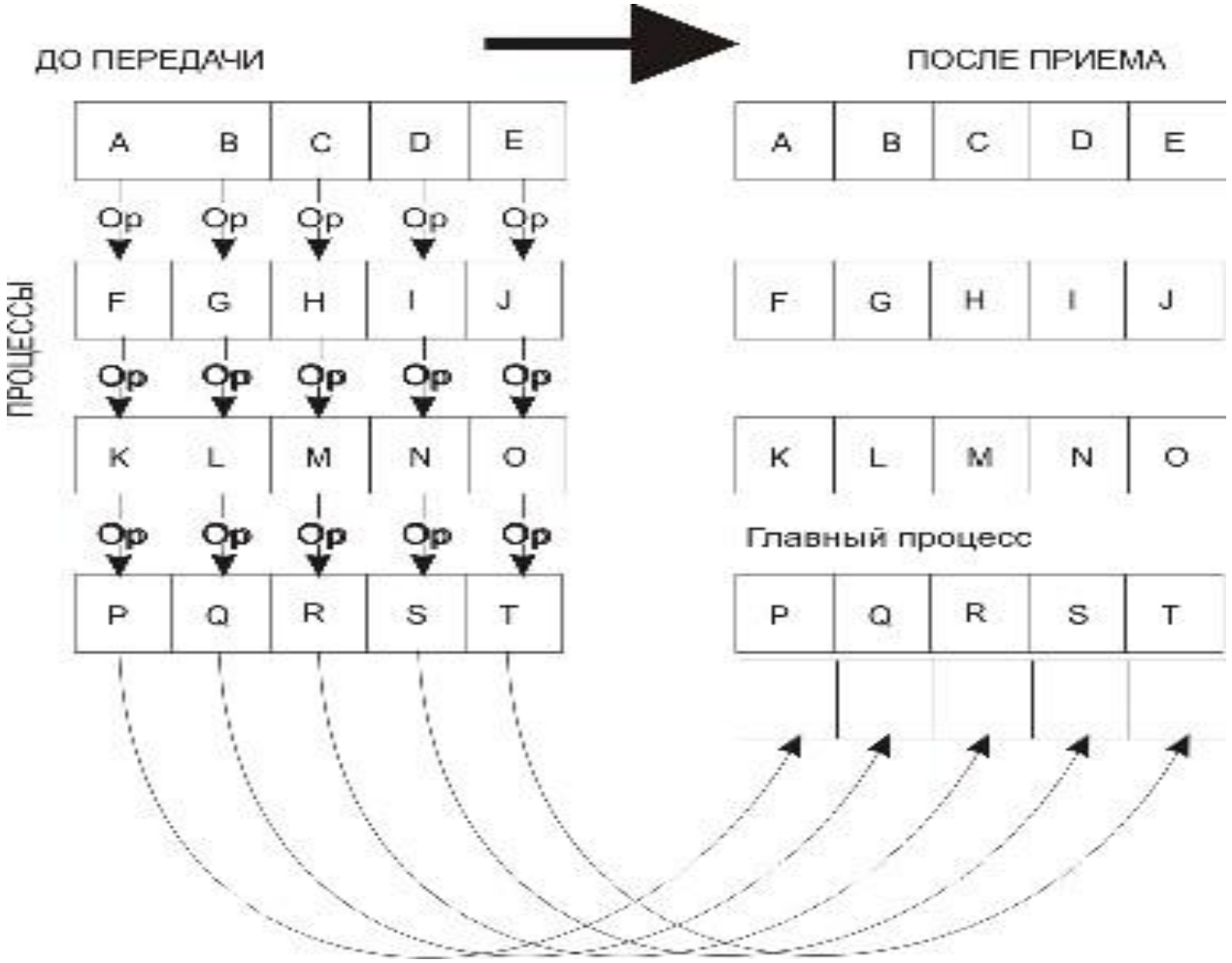
```
int MPI_Reduce(void *buf, void *result, int count,
MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

MPI_Reduce(buf, result, count, datatype, op, root, comm,
ierr)
```

Входные параметры:

- `buf` - адрес буфера передачи;
- `count` - количество элементов в буфере передачи;
- `datatype` - тип данных в буфере передачи;
- `op` - операция приведения;
- `root` - ранг главного процесса;
- `comm` - коммуникатор.

Коллективные обмены

`MPI_Reduce` применяет операцию приведения к операндам из `buf`, а результат каждой операции помещается в буфер результата `result`. `MPI_Reduce` должна вызываться всеми процессами в коммуникаторе `comm`, а аргументы `count`, `datatype` и `op` в этих вызовах должны совпадать.



Коллективные обмены

Пример 1 использования операции редукции

В этой программе сначала создается подгруппа, состоящая из процессов с рангами 1, 3, 5 и 7 (запускать ее на выполнение надо не менее чем в восьми процессах), и соответствующий ей коммуникатор. Редукция выполняется только процессами из этой группы. В конце программы все созданные в процессе ее работы описатели должны быть удалены.

Коллективные обмены

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int myrank, i;
 int count = 5, root = 1;
 MPI_Group MPI_GROUP_WORLD, subgroup;
 int ranks[4] = {1, 3, 5, 7};
 MPI_Comm subcomm;
 int sendbuf[5] = {1, 2, 3, 4, 5};
 int recvbuf[5];

 MPI_Init(&argc, &argv);
 MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
 MPI_Group_incl(MPI_GROUP_WORLD, 4, ranks, &subgroup);
 MPI_Group_rank(subgroup, &myrank);
 MPI_Comm_create(MPI_COMM_WORLD, subgroup, &subcomm);
```

Коллективные обмены

```
if(myrank != MPI_UNDEFINED)
{
MPI_Reduce(&sendbuf, &recvbuf, count, MPI_INT, MPI_SUM, root, subcomm);

if(myrank == root) {
printf("Reduced values");
for(i = 0; i < count; i++){
printf(" %i ", recvbuf[i]);}
}
printf("\n");

MPI_Comm_free(&subcomm);
MPI_Group_free(&MPI_GROUP_WORLD);
MPI_Group_free(&subgroup);
}
 MPI_Finalize();
 return 0;
}
```

Коллективные обмены

Предопределенные операции приведения

Операция	Описание
MPI_MAX	Определение максимальных значений элементов одномерных массивов целого или вещественного типа
MPI_MIN	Определение минимальных значений элементов одномерных массивов целого или вещественного типа
MPI_SUM	Вычисление суммы элементов одномерных массивов целого, вещественного или комплексного типа
MPI_PROD	Вычисление поэлементного произведения одномерных массивов целого, вещественного или комплексного типа
MPI_LAND	Логическое "И"
MPI_BAND	Битовое "И"
MPI_LOR	Логическое "ИЛИ"
MPI_BOR	Битовое "ИЛИ"
MPI_LXOR	Логическое исключающее "ИЛИ"
MPI_BXOR	Битовое исключающее "ИЛИ"
MPI_MAXLOC	Максимальные значения элементов одномерных массивов и их индексы
MPI_MINLOC	Минимальные значения элементов одномерных массивов и их индексы

Коллективные обмены

Допускается определение собственных операций приведения. Для этого используется подпрограмма:

```
int MPI_Op_create(MPI_User_function *function, int commute,
MPI_Op *op)

MPI_Op_create(function, commute, op, ierr)
```

Входные параметры:

- `function` - пользовательская функция;
- `commute` - флаг, которому присваивается значение «истина», если операция коммутативна (результат не зависит от порядка операндов).

Коллективные обмены

Описание типа пользовательской функции выглядит следующим образом:

```
typedef void (MPI_User_function)(void *a, void *b, int *len,
MPI_Datatype *dtype)
```

Здесь операция определяется так:

```
b[I] = a[I] op b[I]
```

для `I = 0, …, len – 1.`

Коллективные обмены

После завершения операций приведения пользовательская функция должна быть удалена.

Удаление пользовательской функции выполняется подпрограммой:

```
int MPI_Op_free(MPI_Op *op)

MPI_Op_free(op, ierr)
```

После завершения вызова `op` присваивается значение `MPI_OP_NULL`.

Коллективные обмены

Пример использования операции приведения: вычисление числа π методом Монте-Карло. Два файла **pi_compute.c** и **mc_trials.c**.

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

float mc_trials(int trials);

main(int argc, char **argv)
{
float homepi, pisum, pi, avepi;
int mytid, nproc, rcode, i;
int trials = 10000, rounds = 20, master = 0;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &mytid);
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
printf ("MPI task ID = %d\n", mytid);
```

Коллективные обмены

```
srandom (mytid);

avepi = 0;
for (i = 0; i < rounds; i++) {
     homepi = dboard(trials);

      rcode = MPI_Reduce(&homepi, &pisum, 1, MPI_FLOAT,
MPI_SUM, master, MPI_COMM_WORLD);
    if (rcode != 0)
              printf("%d: failure on MPI_Reduce\n", mytid);

     if (mytid == master) {
          pi = pisum/nproc;
          avepi = ((avepi * i) + pi)/(i + 1);
          printf("   After %3d throws, average value of pi =
%10.8f\n", (trials * (i + 1)),avepi);
          }
    }
    MPI_Finalize();
}
```

Коллективные обмены

```
#include <stdlib.h>
#define sqr(x) ((x)*(x))

float mc_trials(int trials)
     {
     double x_coord, y_coord, pi, r;
     int score, n;
     unsigned long cconst = 2147483647.;
     score = 0;
     for (n = 1; n <= darts; n++)  {
          r = (float)rand() / cconst;
          x_coord = (2.0 * r) - 1.0;
          r = (float)rand() / cconst;
          y_coord = (2.0 * r) - 1.0;
          if ((sqr(x_coord) + sqr(y_coord)) <= 1.0)
               score++;
          }

    pi = 4.0 * (float)score/trials;
    return(pi);
     }
```

Коллективные обмены

```
pi = 4.0 * (double)score/(double)trials;
     return(pi);
     }
```

Коллективные обмены

Операция сканирования

Операции *сканирования* (частичной редукции) выполняются следующей подпрограммой:

```
int MPI_Scan(void *sendbuf, void *rcvbuf, int count,
MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

MPI_Scan(sendbuf, rcvbuf, count, datatype, op, comm, ierr)
```

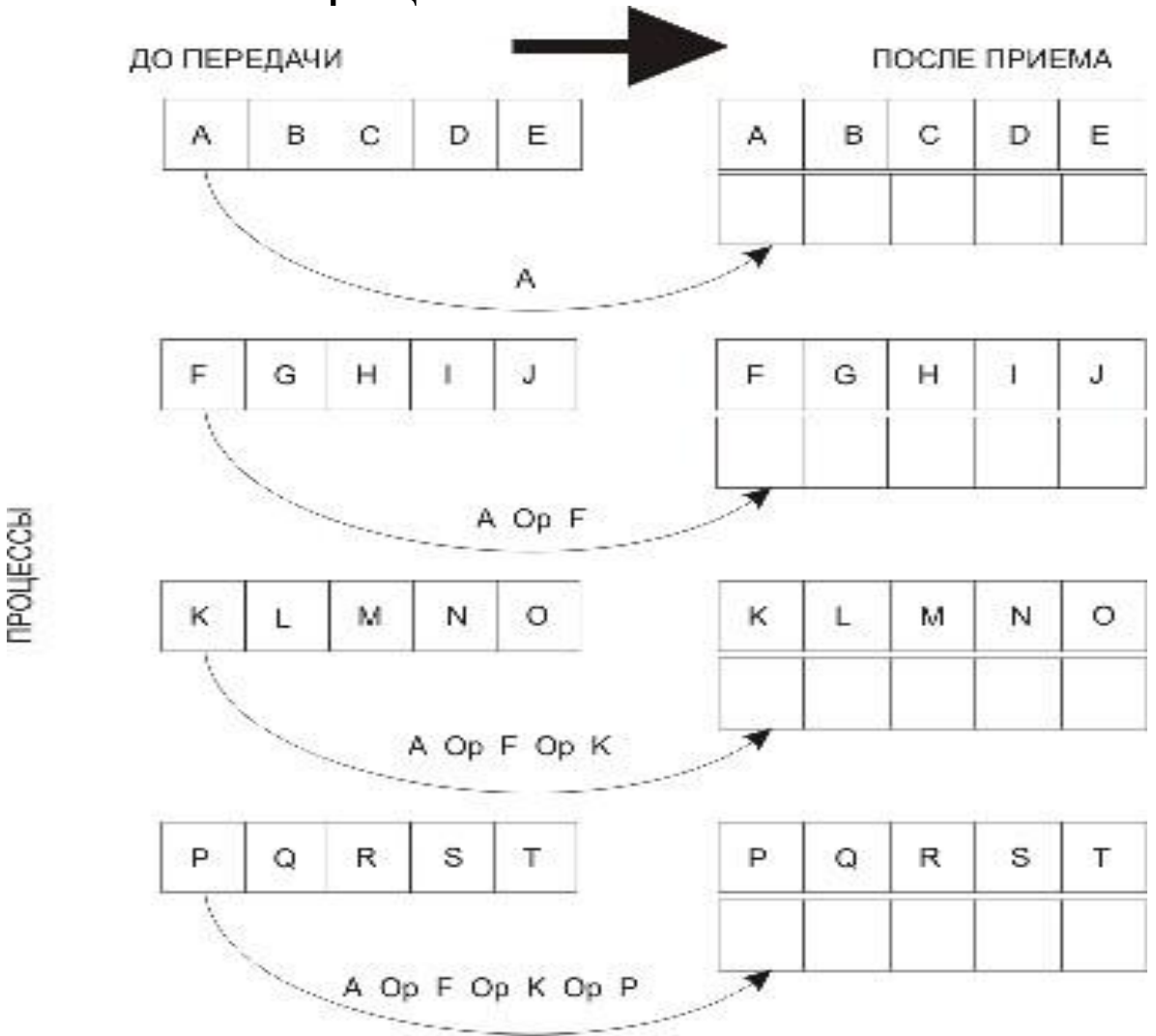
Входные параметры:

- `sendbuf` - начальный адрес буфера передачи;
- `count` - количество элементов во входном буфере;
- `datatype` - тип данных во входном буфере;
- `op` - операция;
- `comm` - коммуникатор.

Выходной параметр: `rcvbuf` - стартовый адрес буфера приема.

Коллективные обмены

При выполнении операции сканирования в буфере приёма процесса с рангом `i` будут содержаться результаты приведения значений в буферах передачи процессов с рангами `0, …, i`. В остальном эта операция аналогична операции `MPI Reduce`.



Коллективные обмены

Векторная операция распределения данных

Векторная подпрограмма распределения данных:

```
int MPI_Scatterv(void *sendbuf, int *sendcounts, int *displs,
MPI_Datatype sendtype, void *rcvbuf, int rcvcount,
MPI_Datatype rcvtype, int root, MPI_Comm comm)

MPI_Scatterv(sendbuf, sendcounts, displs, sendtype, rcvbuf,
rcvcount, rcvtype, root, comm, ierr)
```

Входные параметры:

- `sendbuf` - адрес буфера передачи;
- `sendcounts` - целочисленный одномерный массив, содержащий количество элементов, передаваемых каждому процессу (индекс равен рангу адресата). Его длина равна количеству процессов в коммуникаторе;

Коллективные обмены

Входные параметры:

- `displs` - целочисленный массив, длина которого равна количеству процессов в коммуникаторе. Элемент с индексом i задает смещение относительно начала буфера передачи. Ранг адресата равен значению индекса i;
- `sendtype` - тип данных в буфере передачи;
- `rcvcount` - количество элементов в буфере приема;
- `rcvtype` - тип данных в буфере приема;
- `root` - ранг передающего процесса;
- `comm` - коммуникатор.

Выходной параметр: `rcvbuf` - адрес буфера приема.

Коллективные обмены

Векторная операция сбора данных

Сбор данных от всех процессов в заданном коммуникаторе и запись их в буфер приема с указанным смещением выполняется подпрограммой *векторного* сбора данных:

```
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_Gatherv(sendbuf, sendcount, sendtype, recvbuf,
recvcounts, displs, recvtype, root, comm, ierr)
```

Список параметров у этой подпрограммы похож на список параметров подпрограммы `MPI_Scatterv`. В обменах, выполняемых подпрограммами `MPI_Allgather` и `MPI_Alltoall`, нет главного процесса. Детали отправки и приема важны для всех процессов, участвующих в обмене.

Коллективные обмены

Пересылка данных по схеме «каждый - всем»

```
int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype,
MPI_Comm comm)

MPI_Alltoall(sendbuf, sendcount, sendtype, rcvbuf, rcvcount,
rcvtype, comm, ierr)
```

Входные параметры:

- `sendbuf` - начальный адрес буфера передачи;
- `sendcount` - количество элементов данных, пересылаемых каждому процессу;
- `sendtype` - тип данных в буфере передачи;
- `rcvcount` - количество элементов данных, принимаемых от каждого процесса;
- `rcvtype` - тип принимаемых данных;
- `comm` - коммуникатор.

Выходной параметр: `rcvbuf` - адрес буфера приема.

Коллективные обмены

Векторными версиями `MPI_Allgather` и `MPI_Alltoall` являются подпрограммы `MPI_Allgatherv` и `MPI_Alltoallv`.
Векторные операции позволяют детализировать процесс коллективного обмена.

Коллективные обмены

Синхронизация

Синхронизация с помощью «барьера» выполняется с помощью подпрограммы:

```
int MPI_Barrier(MPI_Comm comm)

MPI_Barrier(comm, ierr)
```

Синхронизация с помощью «барьера» - простейшая форма синхронизации коллективных обменов. Она не требует пересылки данных. Обращение к подпрограмме `MPI_Barrier` блокирует выполнение каждого процесса из коммуникатора `comm` до тех пор, пока все процессы не вызовут эту подпрограмму, таким образом, «толщина барьера» здесь максимальная – она равна числу процессов в указанном коммуникаторе.
Барьерная синхронизация относится к числу коллективных операций потому, что выполнить соответствующий вызов должны все процессы.

Заключение

В этой лекции мы рассмотрели:

- особенности и свойства коллективных обменов;
- различные операции коллективного обмена – широковещательную рассылку, сбор и распределение данных, приведение и сканирование и т. д.;
- синхронизацию при организации коллективных обменов.

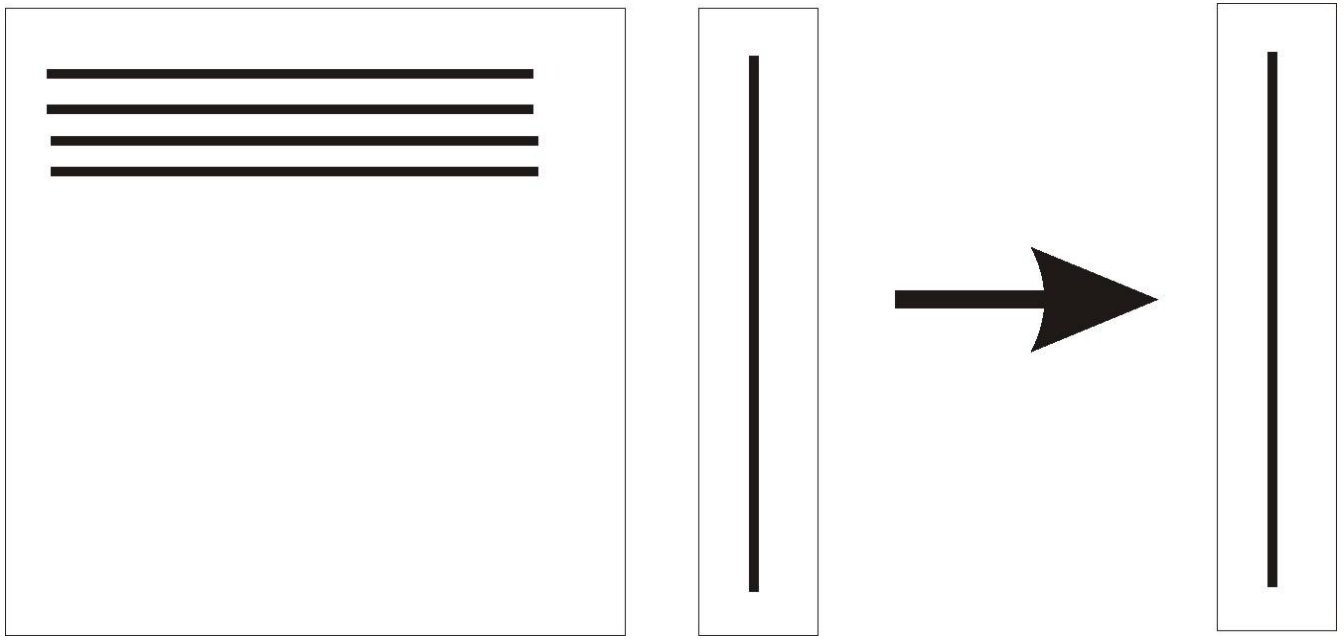
Задания для самостоятельной работы

Решения следует высылать по электронной почте:

parallel-g112@yandex.ru

Задания для самостоятельной работы

Составьте алгоритм и напишите параллельную программу вычисления произведения матрицы на вектор. На приведенной схеме приведен предлагаемый способ декомпозиции. Используйте операции коллективного обмена для пересылки всем процессам вектора.



Задания для самостоятельной работы

Напишите параллельную программу вычисления произведения матрицы на матрицу. Используйте ленточную декомпозицию и операции коллективного обмена.

Тема следующей лекции

Группы процессов и коммуникаторы