Летняя суперкомпьютерная академия МГУ,
22 июня – 1 июля 2015 г

Трек
Информатика в школе

Лекция

Введение в технологию параллельного программирования MPI

Лектор: доцент Н.Н.Попова,

26 июня 2015 г.

План

- Модели параллельного программирования
- Организация взаимодействия параллельных процессов в модели MPI
- Основные функции передачи сообщений между 2 процессами MPI
- Функции MPI для коллективной передачи сообщений
- Материалы лекции:
  angel.cs.msu.su/~popova/SSA

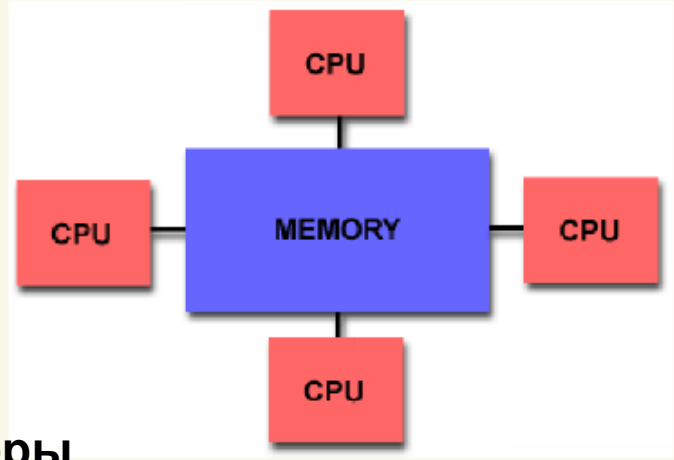
Базовые основы построения параллельных вычислительных систем.

2 основных класса параллельных ВС:

- **системы с общей памятью** (многоядерные процессоры, специальные SMP-процессоры)

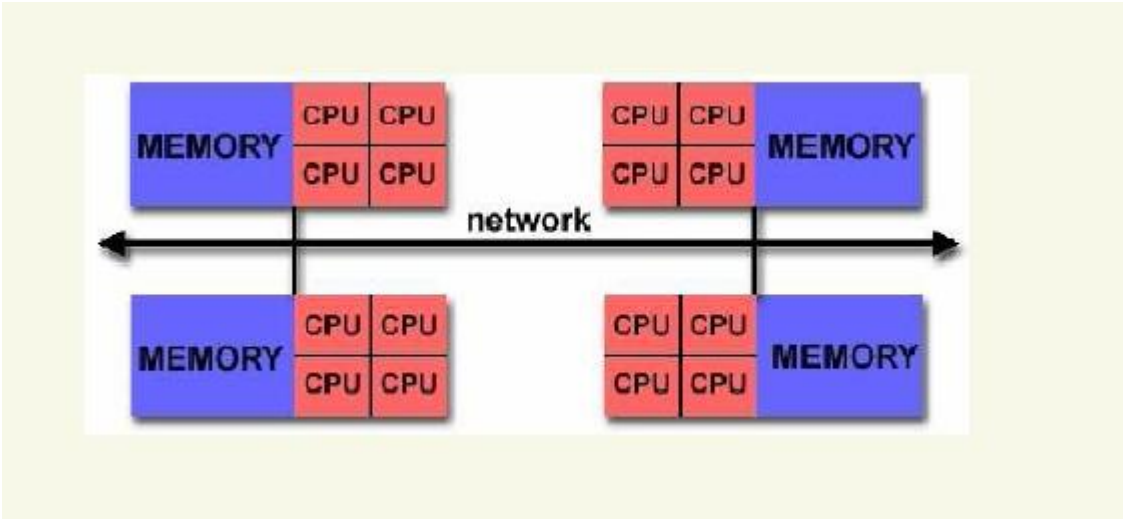
- **системы с распределенной памятью**

Системы с общей памятью



Примеры:
-Многоядерные процессоры
-SMP - системы

Системы с распределенной памятью



Параллельная программа

- ***Параллельная программа*** – программа, в которой явно определено параллельное выполнение всей программы либо ее фрагментов (блоков, операторов, инструкций). Программу, в которой параллелизм поддерживается ***неявно***, не будем относить к параллельным.
- ***Процесс*** – программа во время выполнения (интуитивно). Существует несколько более формальных определений

- Параллельная программа, как правило, выполняется в рамках ***нескольких процессов, ВЗАИМОДЕЙСТВУЩИХ!***
- ***Поток*** – легковесный процесс. В рамках одного процесса может существовать ***НЕСКОЛЬКО ПОТОКОВ*** – модель OpenMP- программ.

Модели параллельных программ

- **Системы с общей памятью**
  - Программирование, основанное на потоках
  - Программа строится на базе последовательной программы
  - Возможно автоматическое распараллеливание компилятором с использованием соответствующего ключа компилятора
  - Директивы компиляторов (OpenMP, …)

- **Системы с распределенной памятью**
  - Программа состоит из параллельных процессов
  - Явное задание коммуникаций между процессами – обмен сообщениями **“Message Passing”**

  Реализация - Message Passing библиотек:
    - MPI (“Message Passing Interface”)
    - PVM (“Parallel Virtual Machine”)
    - Shmem

Пример параллельной программы программы (C, OpenMP)

Сумма элементов массива

```
#include <stdio.h>
#define N 1024
int main()
{ double sum;
  double a[N];
  int status, i, n =N;
  for (i=0; i<n; i++){
    a[i] = i*0.5;  }
 sum =0;
#pragma omp for reduction (+:sum)
 for (i=0; i<n; i++)
  sum = sum+a[i];
printf (“Sum=%f\n”, sum);
}
```

MPI

```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, all_sum;
  double a[N];
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
```

```
n= n/ size;
  for (i=rank*n;  i<n; i++){
    a[i] = i*0.5;  }

 sum =0;
for (i=0; i<n; i++)
  sum = sum+a[i];
MPI_Reduce(& sum,& all_sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf ("Sum =%f\n", all_sum);
MPI_Finalize();
return 0;
}
```

MPI – стандарт (формальная спецификация)

- MPI 1.1 Standard разрабатывался 92-94
- MPI 2.0 - 95-97
- MPI 2.1 - 2008
- MPI 3.0 – 2012
- Стандарты
  - **http://www.mcs.anl.gov/mpi**
  - **http://www.mpi-forum.org/docs/docs.html**

  Описание функций
  - **http://www-unix.mcs.anl.gov/mpi/www/**

Цель MPI

- Основная цель:
  - Обеспечение переносимости исходных кодов
  - Эффективная реализация
- Кроме того:
  - Большая функциональность
  - Поддержка неоднородных параллельных архитектур

Реализации MPI - библиотеки

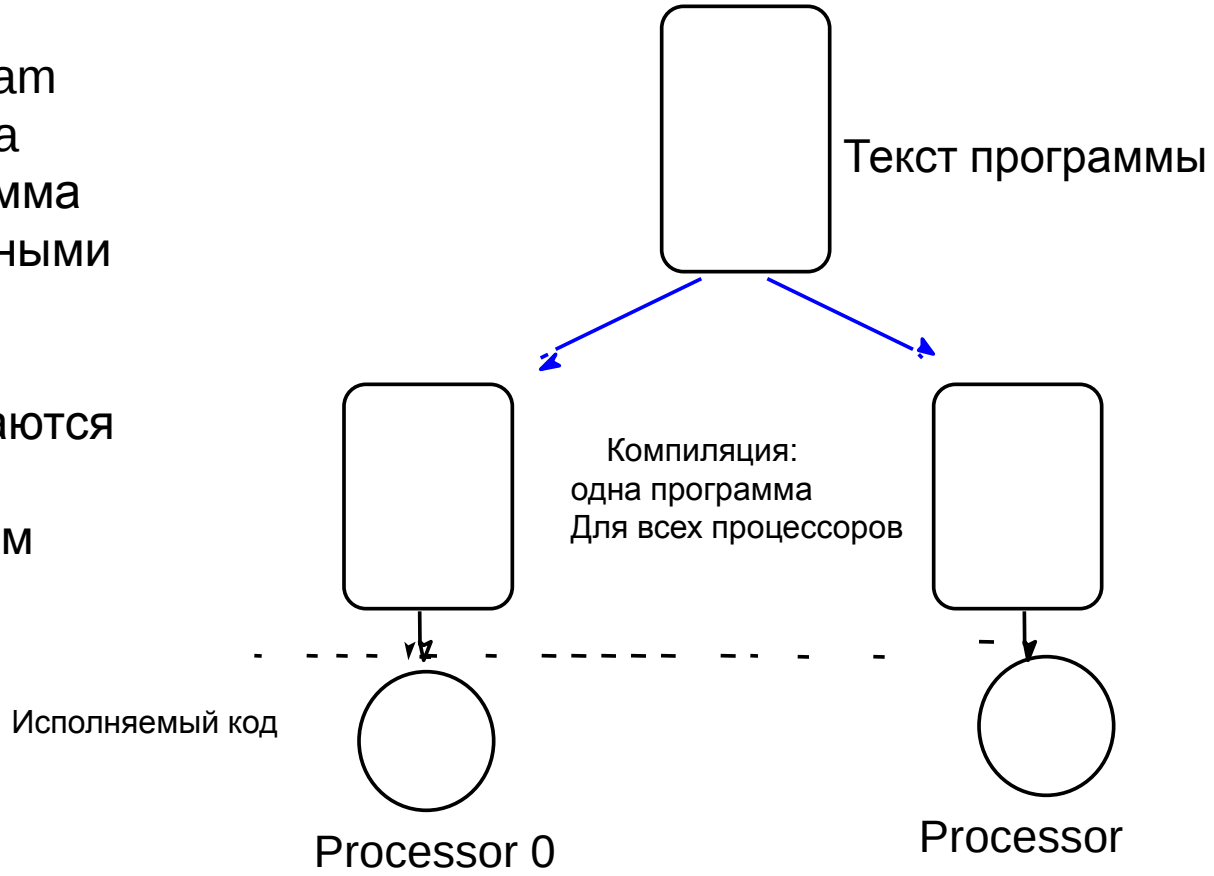
- MPICH
- LAM/MPI
- Mvapich
- OpenMPI
- Коммерческие реализации Intel,IBM и др.

Модель MPI

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

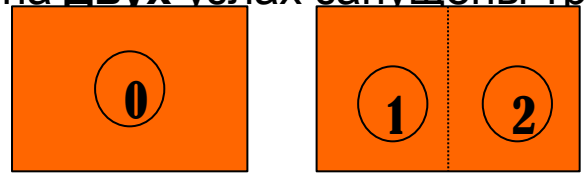
Модель MPI-программ

- **SPMD** – Single Program Multiple Data
- Одна и та же программа выполняется различными процессорами
- Управляющими операторами выбираются различные части программы на каждом процессоре.



Модель выполнения MPI- программы

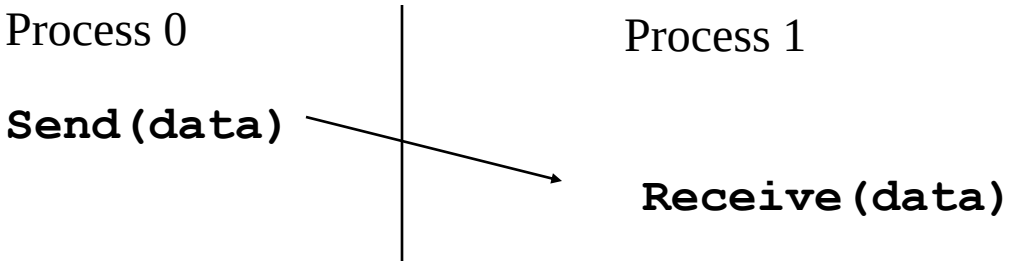
- Запуск: ***mpirun***
- При запуске указываем число требуемых процессоров **np** и название программы: пример: ***mpirun –np 3 prog***
- На выделенных узлах запускается ***np*** копий (процессов) указанной программы
  - Например, на **двух** узлах запущены три копии программы.



- Каждый процесс MPI-программы получает два значения:
  - ***np –*** *число процессов*
  - ***rank*** из диапазона [0 … ***np-1***] – номер процесса
- Любые два процесса могут непосредственно обмениваться данными с помощью функций передачи сообщений

Основы передачи данных в MPI

- Данные посылаются одним процессом и принимаются другим.
- Передача и синхронизация совмещены.



Основы передачи данных в MPI

- Требуется уточнить:
  - Как должны быть описаны данные ?
  - Как должны идентифицироваться процессы?
  - Как получатель получит информацию о сообщении?
  - Что значить завершение передачи?

6 основных функций MPI

- **Как стартовать/завершить параллельное выполнение**
  - MPI_Init
  - MPI_Finalize
- **Кто я (и другие процессы), сколько нас**
  - MPI_Comm_rank
  - MPI_Comm_size
- **Как передать сообщение коллеге (другому процессу)**
  - MPI_Send
  - MPI_Recv

Основные понятия MPI

- Процессы объединяются в ***группы***.
- Группе приписывается ряд свойств (как связаны друг с другом и некоторые другие). Получаем ***коммуникаторы***
- Процесс идентифицируется своим номером в группе, привязанной к конкретному коммуникатору.
- При запуске параллельной программы создается специальный коммуникатор с именем ***MPI_COMM_WORLD***
- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.

Понятие коммуникатора MPI

- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.
- Наиболее часто используемый коммуникатор `MPI_COMM_WORLD`
  - определяется при вызове `MPI_Init`
  - содержит ВСЕ процессы программы

Типы данных MPI

- Данные в сообщении описываются тройкой: (***address, count, datatype***)
- ***datatype*** (типы данных MPI)

Signed
- MPI_CHAR
- MPI_SHORT
- MPI_INT
- MPI_LONG

Unsigned
- MPI_UNSIGNED_CHAR
- MPI_UNSIGNED_SHORT
- MPI_UNSIGNED
- MPI_UNSIGNED_LONG

- MPI_FLOAT
- MPI_DOUBLE
- MPI_LONG_DOUBLE

Базовые MPI-типы данных (C)

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Специальные типы MPI

- MPI_Comm
- MPI_Status
- MPI_datatype

Понятие тэга

- Сообщение сопровождается определяемым пользователем признаком – целым числом – ***тэгом*** для идентификации принимаемого сообщения
- Теги сообщений у отправителя и получателя должны быть согласованы. Можно указать в качестве значения тэга константу **`MPI_ANY_TAG`**.
- Некоторые не-MPI системы передачи сообщений называют тэг типом сообщения. MPI вводит понятие тэга, чтобы не путать это понятие с типом данных MPI.

Структура MPI программ

MPI Include File

Инициализация MPI

Вычисления, обмен сообщениями

Завершение MPI

C: MPI helloworld.c

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    printf("Hello, MPI world\n");
    MPI_Finalize();
    return 0; }
```

Формат MPI-функций

```
error = MPI_Xxxxx(parameter,...);
MPI_Xxxxx(parameter,...);
```

Обработка ошибок MPI-функций

Определяется константой ***MPI_SUCCESS***

```
I
int error;
……
error = MPI_Init(&argc, &argv));
If (error != MPI_SUCCESS)
{
 fprintf (stderr, “ MPI_Init error \n”);
return 1;

}
```

Основные группы функций MPI

- Определение среды
- Передачи «точка-точка»
- Коллективные операции
- Производные типы данных
- Группы процессов
- Виртуальные топологии
- Односторонние передачи данных
- Параллельный ввод-вывод
- Динамическое создание процессов
- Средства профилирования

Инициализация MPI

MPI_Init должна первым вызовом, вызывается только один раз

```
int MPI_Init(int *argc, char ***argv)
```

MPI_Comm_size
Количество процессов в коммуникаторе

- **Размер коммуникатора**

```
int MPI_Comm_size(MPI_Comm comm,
                  int *size)
```

IN comm - коммуникатор

OUT size - размер коммуникатора - РЕЗУЛЬТАТ

MPI_Comm_rank
номер процесса (process rank)

- Номер процесса в коммуникаторе
  - Начинается с 0 до (*n*-1), где *n* – число процессов
- Используется для определения номера процесса-отправителя и получателя

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Результат – rank -номер процесса

Завершение MPI-процессов

- Никаких вызовов MPI функций после

C:

int MPI_Finalize()

int MPI_Abort (MPI_Comm_size(MPI_Comm comm, int*errorcode)

Если какой-либо из процессов не выполняет MPI_Finalize, программа зависает.

Hello, MPI world! (2)

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
   int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, MPI world! I am %d of %d\n“,rank,size);
    MPI_Finalize();
    return 0; }
```

Трансляция MPI-программ

- Трансляция

  mpicc –o <имя_программы> <имя>.c <опции>

  Например:

  mpicc –o hw helloworld.c
- Запуск в интерактивном режиме

  mpirun –np 128 hw
- Запуск в пакетном режиме на Ломоносове
- Запуск в пакетном режиме на Blue Gene/P

Взаимодействие «точка-точка»

- Самая простая форма обмена сообщением
- Один процесс посылает сообщения другому
- Несколько вариантов реализации того, как пересылка и выполнение программы совмещаются

Функции MPI передачи «точка-точка»

Point-to-Point Communication Routines		
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Start	MPI_Startall
MPI_Test	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome

Передача сообщений типа «точка-точка»

- Взаимодействие между двумя процессами
- Процесс-отправитель(Source process) ***посылает*** сообщение процессу-получателю (Destination process)
- Процесс-получатель ***принимает*** сообщение
- Передача сообщения происходит в рамках заданного коммуникатора
- Процесс-получатель определяется рангом в коммуникаторе

Завершение

- “Завершение” передачи означает, что буфер в памяти, занятый для передачи, может быть безопасно использован для доступа, т.е.
  - Send: переменная, задействованная в передаче сообщения, может быть доступна для дальнейшей работы
  - Receive: переменная, получающая значение в результате передачи, может быть использована

Функция передачи сообщения MPI_Send

*int **MPI_Send**(void *buf, int count, MPI_Datatype datatype,int dest, int tag, MPI_Comm comm)*

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
dest	-	rank процесса-получателя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор

Пример:

`MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)`:

Передача массива *data, 500 элементов вещественного типа процессу с номером 6, тег сообщения 33, коммуникатор MPI_COMM_WORLD*

Функция приема сообщения MPI_Recv

*int MPI_Recv(void *buf,int count, MPI_Datatype datatype,int source, int tag, MPI_Comm comm,* ***MPI_Status *status*** *)*

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
source	-	rank процесса-отправителя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор,
status	-	статус

Пример:

```
MPI_Recv(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD,&stat)
```

Пример: MPI Send/Receive (1)

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[]){
int numtasks, rank, dest, source, rc, tag=1;
char inmsg, outmsg='X';
MPI_Status Stat;
MPI_Init (&argc,&argv);
MPI_Comm_size (MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
if (rank == 0) {
  dest = 1;
rc = MPI_Send (&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  printf("Rank0 sent: %c\n", outmsg);
source = 1;
  rc = MPI_Recv (&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD,
      &Stat);  }
```

Пример: MPI Send/Receive (2)

```
else if (rank == 1) {
  source = 0;
  rc = MPI_Recv (&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  printf("Rank1 received: %c\n", inmsg);
 dest = 0;
  rc = MPI_Send (&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }

MPI_Finalize();
}
```

Wildcarding (джокеры)

- Получатель может использовать джокер
  для получения сообщения от **ЛЮБОГО** процесса
  `MPI_ANY_SOURCE`

- Для получения сообщения с ЛЮБЫМ тэгом
  `MPI_ANY_TAG`

- Реальные номер процесса-отправителя и тэг возвращаются через параметр ***`status`***

Информация о завершившемся приеме сообщения

- Возвращается функцией **`MPI_Recv`** через параметр **`status`**

- Содержит:
  - Source: ***`status.MPI_SOURCE`***
  - Tag: ***`status.MPI_TAG`***
  - Count: ***`MPI_Get_count`***

Определение размера полученного сообщения

- Может быть меньшего размера, чем указано в функции MPI_Recv
- `count` – число реально полученных элементов

C:

```
int MPI_Get_count  (MPI_Status *status,
        MPI_Datatype datatype, int *count)
```

Пример

```
int recvd_tag, recvd_from, recvd_count;
MPI_Status status;
MPI_Recv (..., MPI_ANY_SOURCE, MPI_ANY_TAG, ..., &status )
recvd_tag  = status.MPI_TAG;
recvd_from = status.MPI_SOURCE;
MPI_Get_count( &status, datatype, &recvd_count );
```

Условия успешного взаимодействия «точка-точка»

- Отправитель должен указать правильный rank получателя
- Получатель должен указать верный rank отправителя
- Одинаковый коммуникатор
- Тэги должны соответствовать друг другу
- Буфер у процесса-получателя должен быть достаточного объема

Замер времени MPI_Wtime

- Время замеряется в секундах
- Выделяется интервал в программе

```
double MPI_Wtime(void)
```

Пример.

```
double start, finish, time ;
start= MPI_Wtime ();
MPI_Send(…);
finish = MPI_Wtime();
time= finish - start;
```

Проверка статуса приема сообщения: MPI_Probe

int MPI_Probe(int *source*, int *tag*, MPI_Comm *comm*,
MPI_Status* *status*)

Параметры аналогичны функции MPI_Recv

Пример (1)

```
if (rank == 0) { // процесс 0
// Процесс 0 посылает  SIZE целых процессу 1
    MPI_Send(buf, size, MPI_INT, 1, 0, MPI_COMM_WORLD);
// В этой точке передача завершена
 printf("0 sent %d numbers to 1\n", size);
 } else if (rank == 1) { // процесс 1
    MPI_Status status;
    // Проверка получеия сообщения
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);
```

Пример (2)

```
/* По возврату можно проверить атрибуты полученного
     сообщения:размер и др.
   Проверим размер сообщения
*/
 MPI_Get_count(&status, MPI_INT, &size)
 /* Получим массив нужного размера в динамической памяти
*/
 int* number_buf = (int*)malloc(sizeof(int) * size);
 // Примем сообщение в этот массив
 MPI_Recv(number_buf, size, MPI_INT, 0, 0, MPI_COMM_WORLD,
          MPI_STATUS_IGNORE);
   printf("1 dynamically received %d numbers from 0.\n",
         number_amount);
   free(number_buf); // Освободим динамическую память
 }
```

Функции неблокирующих передач

```
MPI_Isend(buf,count,datatype,dest,tag,comm,request)
```

```
MPI_Irecv(buf,count,datatype,source,tag,comm, request)
```

Проверка завершения операций `MPI_Wait()` and `MPI_Test()`.

`MPI_Wait()` ожидание завершения.

`MPI_Test()` проверка завершения. Возвращается флаг, указывающий на результат завершения.

Пример использования неблокирующих передач

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* find rank */
if (myrank == 0) {// процесс 0
   int x;
   MPI_Isend(&x,1,MPI_INT, 1, msgtag, MPI_COMM_WORLD,
   &req1);
   compute();// Проводим вычисления
   MPI_Wait(&req1, &status);// Ожидаем завершения передачи
} else if (myrank == 1) {// процесс 1
   int x;
   MPI_Recv(&x,1,MPI_INT,0,msgtag, MPI_COMM_WORLD, &status);
}
```

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются ВСЕМИ процессами в коммуникаторе
- Примеры:
  - Broadcast, scatter, gather (рассылка данных)
  - Global sum, global maximum, и т.д. (Коллективные операции)
  - Барьерная синхронизация

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)
- Нет неблокирующих коллективных операций
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера

Барьерная синхронизация

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

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

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
  ```

Scatter

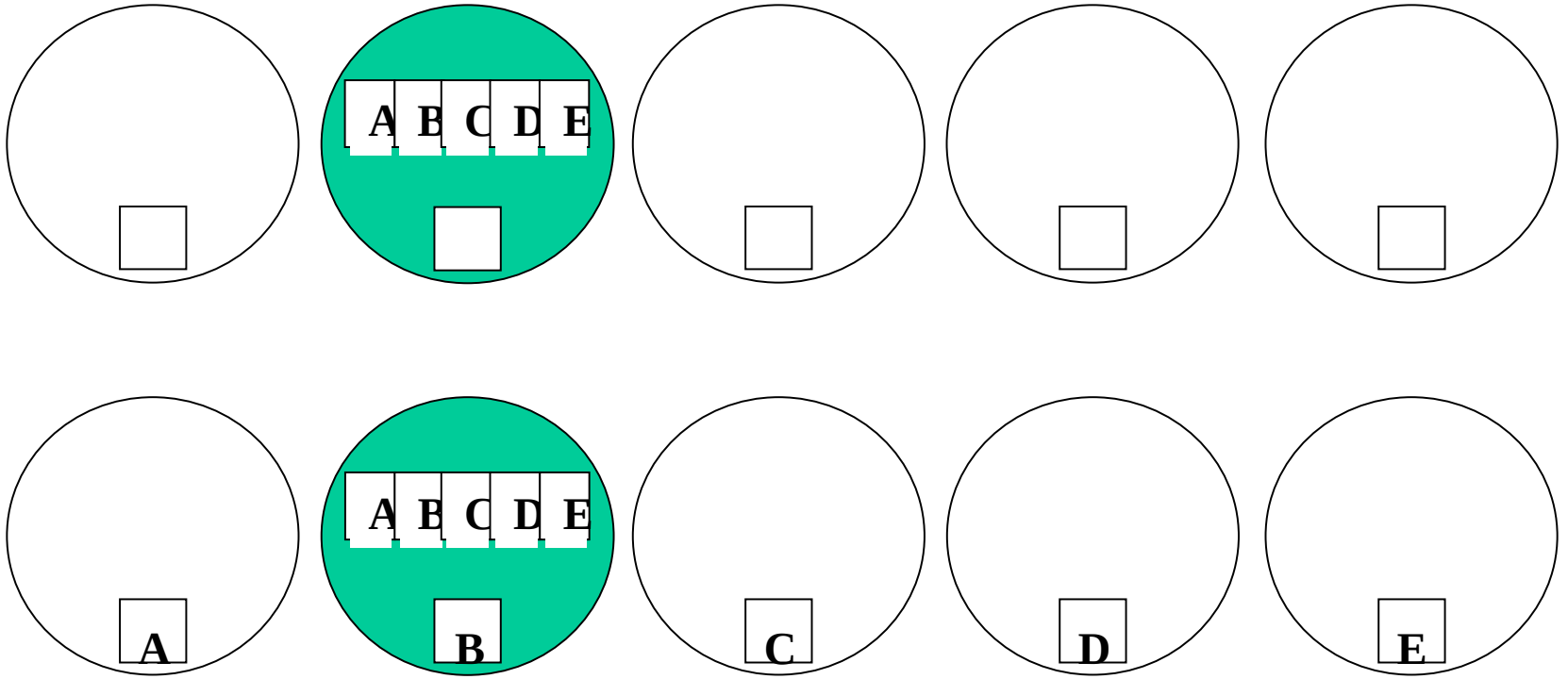
- One-to-all communication: различные данные из одного процесса рассылаются всем процессам коммуникатора (в порядке их номеров)

```
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 имеют смысл только для процесса root

Scatter – графическая иллюстрация

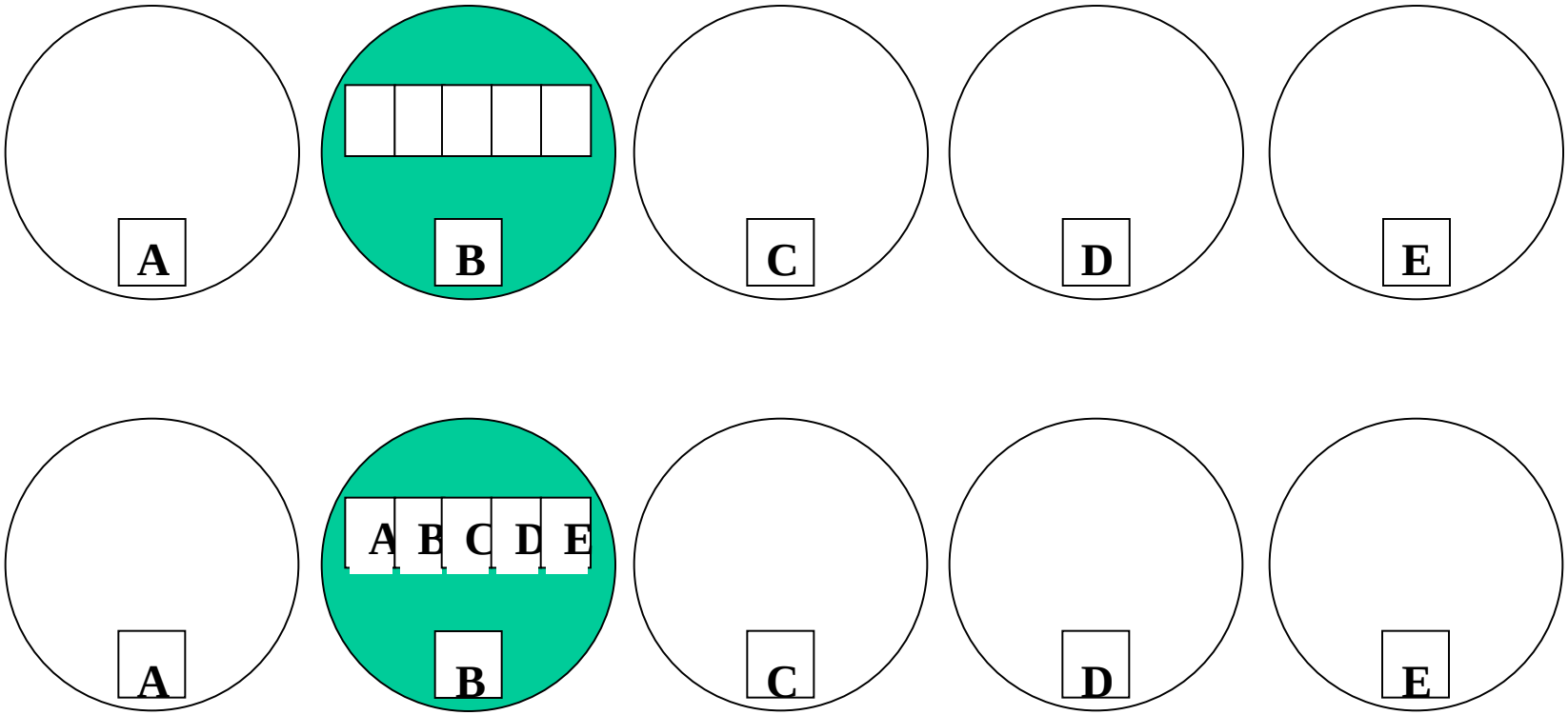


Gather

- All-to-one передачи: различные данные 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 илюстрация



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)
```

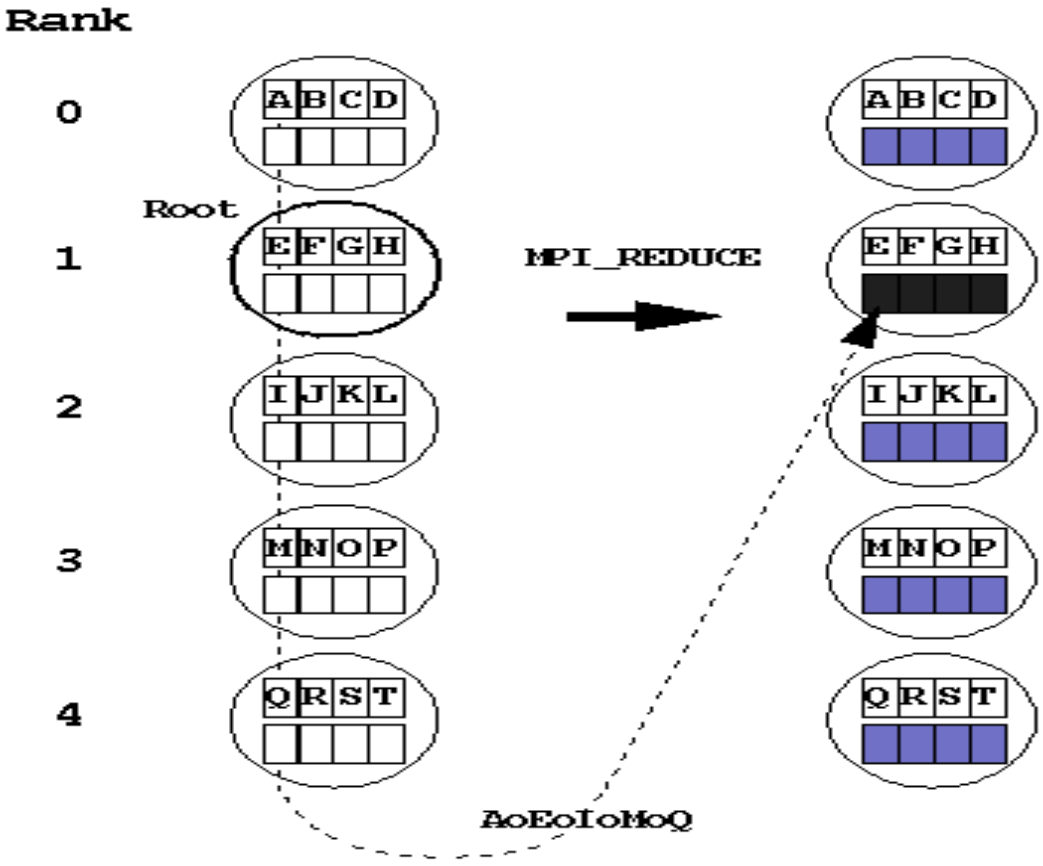
- **count** число операций “*op*” выполняемых над последовательными элементами буфера **sendbuf**
- (также размер **recvbuf**)

- **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

MPI_Reduce


Rank
0
1
2
3
4
Root
A B C D
E F G H
I J K L
M N O P
Q R S T
MPI_REDUCE
A B C D
E F G H
I J K L
M N O P
Q R S T
AoEoIoMoQ