Коллективные функции.

В операциях коллективного взаимодействия процессов участвуют все процессы коммуникатора. Коллективные функции блокирующие. Такие функции не используют идентификаторы и теги, однако дается гарантия, что сообщения не будут пересекаться с обычными приемом и передачей.

Коллективные функции.

Процедура `int MPI_Barrier( MPI_Comm comm)`

Барьер блокирует работу процессов до тех пор, пока все остальные процессы коммуникатора `comm` не выполнят эту процедуру. Все процессы должны вызвать `MPI_Barrier,` хотя и может быть для разных процессов в место вызова может быть разным. Все барьеры равнозначные.

Коллективные функции.

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

#define NTIMES 1000000

int main(int argc, char **argv)
{
    int rank, size;
    double time_start, time_finish;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int *ibuf = (int*)malloc(size * NTIMES * sizeof(int));

    time_start = MPI_Wtime();

    for(int i = 0 ; i < rank * NTIMES ; i ++  ){
        ibuf[i] = i * i;
    }

    MPI_Barrier(MPI_COMM_WORLD);

    printf("rank = %d time = %lf\n", rank, MPI_Wtime()-time_start);
    free(ibuf);

    MPI_Finalize();
}
```

Коллективные функции.

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

Производит рассылку count элементов данных типа `datatype` из массива `buf` от процесса root всем процессам данного коммуникатора `comm`, включая сам рассылающий процесс. Для выполнения необходимо, чтобы `MPI_Bcast` вызывали все процессы.

Коллективные функции.

Для пересылки от процесса `2` всем остальным процессам приложения массива `buf` из `100` целочисленных элементов, нужно, чтобы во всех процессах встретился следующий вызов:

```
MPI_Bcast(buf, 100, MPI_INT,
2, MPI_COMM_WORLD);
```

Коллективные функции.

```
# include <mpi.h>
# include <stdio.h>
# include <string.h>

int main(int argc, char** argv)
{
  int numtasks, rank, root;
  char sbuf[20];
  strcpy(sbuf, "I am not root\0");
  root = 1;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if(rank == root) strcpy(sbuf, "Hello from root\0");
  MPI_Bcast(sbuf, 16, MPI_CHAR, root, MPI_COMM_WORLD);
  if(rank == root) strcpy(sbuf, "I am root\0");

  printf("I am %d. Message received: %s\n", rank, sbuf);
  MPI_Finalize();
  return 0;
}
```

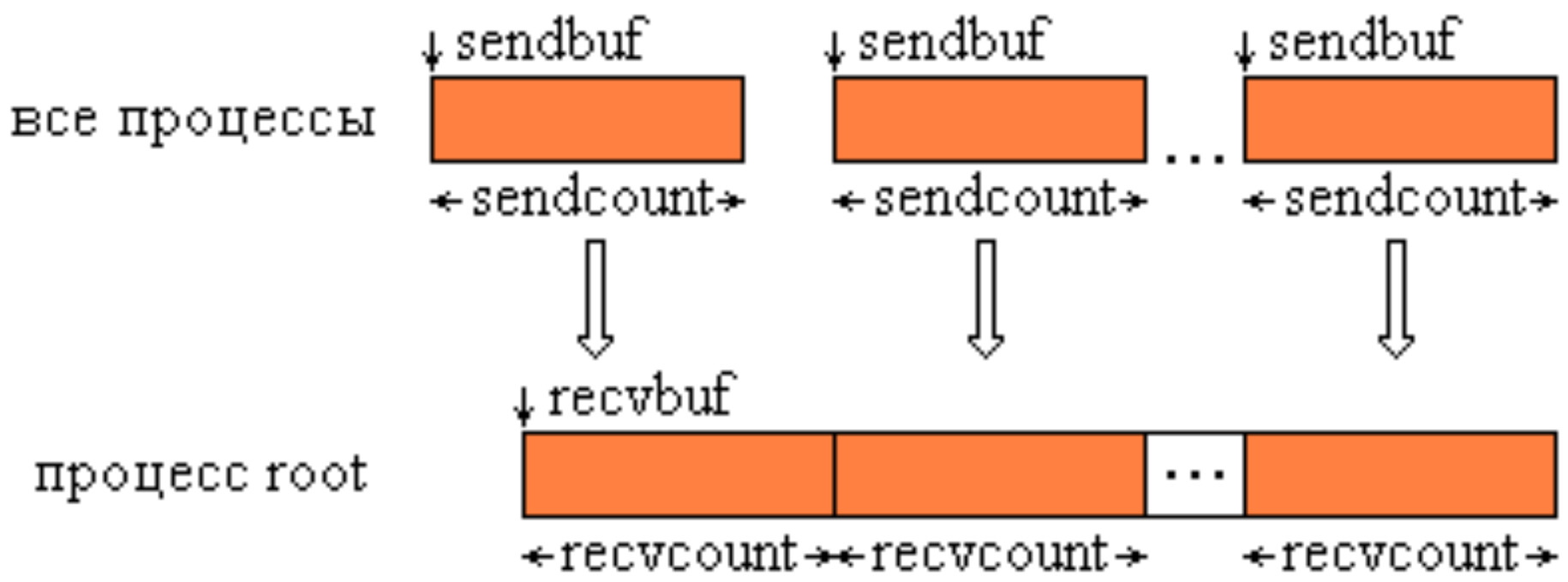
Коллективные функции.

Процедура `MPI_Gather(void * sbuf, int scount, MPI_Datatype stype, void * rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)`

Собирает данные из `scount` элементов типа `stype` из массива `sbuf` со всех процессов коммуникатора `comm` в буфер `rbuf` процесса `root`. Данные сохраняются в `rbuf` в порядке возрастания номеров процессов. Если для процесса root прием должен быть произведен в один и тот же буфер, то на месте `sbuf` можно указать значение `MPI_IN_PLACE`.

Коллективные функции.

Тип посылаемых элементов должен совпадать с типом получаемых элементов, а число отправляемых данных должно равняться числу принимаемых данных. То есть, rcount в вызове из процесса root - это число собираемых от каждого процесса элементов, а не общее количество собранных элементов.



Коллективные функции.

На процессе `root` существенными являются значения всех параметров, а на остальных процессах - только значения параметров `sbuf, scount, stype, root` и comm. Значения параметров root и comm должны быть одинаковыми у всех процессов. Параметр `rcount` у процесса root обозначает число элементов типа `rtype`, принимаемых от каждого процесса.

Коллективные функции.

Если для посылки и приема данных должен использоваться один и тот же буфер, то на месте аргумента `sbuf` процесса root можно указать значение `MPI_IN_PLACE`. В этом случае аргументы `scount` и `stype` игнорируются, и предполагается, что порция данных процесса root уже расположена в соответствующем месте буфера приема `rbuf`.

Коллективные функции.

Например, чтобы процесс `2` собрал в массив `rbuf` по `10` целочисленных элементов массивов `buf` со всех процессов приложения, нужно, чтобы во всех процессах встретился следующий вызов:

```
MPI_Gather(buf, 10, MPI_INT,
rbuf, 10, MPI_INT, 2,
MPI_COMM_WORLD);
```

Коллективные функции.

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

int main(int argv, char **argc){
    int rank, size;
    MPI_Init(&argv, &argc);
    MPI_Comm_rank( MPI_COMM_WORLD, &rank);
    MPI_Comm_size( MPI_COMM_WORLD, &size);
    int recv_buffer[3*size];
    if(rank == 0){
        int buffer[3] = {0, 1, 2};
        MPI_Gather(buffer, 3, MPI_INT, recv_buffer, 3, MPI_INT, 0,
MPI_COMM_WORLD);
        for(int i = 0; i < 3*size; ++i){
           printf("%d ", recv_buffer[i]);
        }
        printf("\n");
    }else{
        int buffer[3] = {1+rank*3, 2+rank*3, 3+rank*3};
        MPI_Gather(buffer, 3, MPI_INT, recv_buffer, 3, MPI_INT, 0,
MPI_COMM_WORLD);
    }

    MPI_Finalize();

    return 0;
}
```

Коллективные функции.

Процедура `int MPI_Gatherv( void * sbuf, int scount, MPI_Datatype stype, void * rbuf, int * rcounts, int * displs, MPI_Datatype retype, int root, MPI_Comm comm)`

Осуществляет сборку различного количества данных из массива `sbuf`. Порядок расположения в массиве данных задает массив `displs`. Количество данных указывается в массиве `rcounts`. `rcounts` представляет собой целочисленный массив, содержащий количество элементов, передаваемых от каждого процесса (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе). `distpls` - целочисленный массив массив, содержит смещения относительно начала массива `rbuf` (индекс равен рангу адресата).

Коллективные функции.

Процедура `int MPI_Gatherv( void * sbuf, int scount, MPI_Datatype stype, void * rbuf, int * rcounts, int * displs, MPI_Datatype retype, int root, MPI_Comm comm)`

`sbuf` - стартовый адрес буфера отправителя,
`scount` - количество элементов в буфере отправки,
`stype` - тип данных элементов буфера отправки,
`rbuf` - стартовый адрес буфера получателя,
`rcount` - массив, содержащий количество элементов, которые будут получены от каждого процесса,
`displs` - массив, задающий смещение относительно `rbuf` (стартового адреса буфера получателя), в который помещаются входящие данные от соответствующего процесса,
`rtype` - тип данных буфера получателя,
`root` - ранг процесса получателя,
`comm` - коммуникатор группы.

Коллективные функции.

```
int main(int argc, char* argv[])
{
    int size;
    int my_rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if(size != 3)
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);

    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    int root_rank = 0;

    switch(my_rank)
    {
        case 0:
        {
            int my_value = 100;
            int counts[3] = {1, 1, 2};
            int displacements[3] = {0, 3, 5};

            int* buffer = (int*)malloc(7, sizeof(int));
            printf("Process %d, my value = %d.\n", my_rank, my_value);
            MPI_Gatherv(&my_value, 1, MPI_INT, buffer, counts, displacements, MPI_INT, root_rank, MPI_COMM_WORLD);
            printf("Values gathered in the buffer on process %d:", my_rank);

            for(int i = 0; i < 7; i++)
                printf(" %d", buffer[i]);

            printf("\n");
            free(buffer);
            break;
    }
    case 1:
    {
        int my_value = 101;

        printf("Process %d, my value = %d.\n", my_rank, my_value);
        MPI_Gatherv(&my_value, 1, MPI_INT, NULL, NULL, NULL, MPI_INT, root_rank, MPI_COMM_WORLD);
        break;
    }
    case 2:
    {
        int my_values[2] = {102, 103};

        printf("Process %d, my values = %d %d.\n", my_rank, my_values[0], my_values[1]);
        MPI_Gatherv(my_values, 2, MPI_INT, NULL, NULL, NULL, MPI_INT, root_rank, MPI_COMM_WORLD);
        break;
    }
}

MPI_Finalize();
```

Коллективные функции.

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

Сборка данных из массивов `sbuf` со всех процессов коммуникатора comm в буфере `rbuf` каждого процесса. Данные

сохраняются в порядке возрастания номеров процессов.

Коллективные функции.

Если для посылки и приема данных должен использоваться один буфер, то на месте аргумента `sbuf` всех процессов можно указать значение `MPI_IN_PLACE`. В этом случае аргументы `scount` и `stype` игнорируются, и предполагается, что порции исходных данных всех процессов уже расположены в соответствующих местах буферов приема `rbuf`.

Коллективные функции.

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

int main(int argv, char **argc){
    int rank, size;
    MPI_Init(&argv, &argc);
    MPI_Comm_rank( MPI_COMM_WORLD, &rank);
    MPI_Comm_size( MPI_COMM_WORLD, &size);
    int recv_buffer[3*size];
    if(rank == 0){
        int buffer[3] = {0, 1, 2};
        MPI_Allgather(buffer, 3, MPI_INT, recv_buffer, 3, MPI_INT, MPI_COMM_WORLD);
        printf("Rank: %d\n", rank);
        for(int i = 0; i < 3*size; ++i){
           printf("%d ", recv_buffer[i]);
        }
        printf("\n");
    }else{
        int buffer[3] = {1+rank*3, 2+rank*3, 3+rank*3};
        MPI_Allgather(buffer, 3, MPI_INT, recv_buffer, 3, MPI_INT, MPI_COMM_WORLD);
        printf("Rank: %d\n", rank);
        for(int i = 0; i < 3*size; ++i){
           printf("%d ", recv_buffer[i]);
        }
        printf("\n");
    }
    MPI_Finalize();
    return 0;
}
```

Коллективные функции.

```
int MPI_Allgatherv(void *sbuf, int
scount, MPI_Datatype stype, void
*rbuf, int *rcounts, int *displs,
MPI_Datatype rtype, MPI_Comm comm)
```

Сборка на всех процессах различного количества данных из `sbuf`. Порядок расположения данных в массиве `rbuf` задаёт массив `displs`.

Коллективные функции.

Процедура `int MPI_Scatter (void * sbuf, int scount, MPI_Datatype stype, void * rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm`)

Производит рассылку по `scount` элементов данных типа `stype` из массива `sbuf` всех процессов коммуникатора comm включая сам процесс отправитель. В отличии от `MPI_Bcast` MPI_`Scatter` подготавливает для каждого процесса свою порцию данных и рассылает её. Вместо аргумента `rbuf` процесса root можно указать значение `MPI_IN_PLACE`, в там случае аргументы `rcount` и `rtype` игнорируются.

На процессе `root` существенными являются значения всех параметров, а на всех остальных процессах — только значения параметров `rbuf`, `rcount`, `rtype`, `source` и `comm`. Значения параметров `source` и `comm` должны быть одинаковыми у всех процессов.

Коллективные функции.

Если для посылки и приема данных должен использоваться один буфер, то на месте аргумента `rbuf` процесса root можно указать значение `MPI_IN_PLACE`. В этом случае аргументы `rcount` и `rtype` игнорируются, и предполагается, что порция данных процесса root не пересылается.

Коллективные функции.

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

#define DATA_SIZE 2

int main(int argc, char *argv[]) {
    int rank, size, root, res;
    MPI_Status status;
    root = 0;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int send[DATA_SIZE * size];
    int count = DATA_SIZE;
    int recv[count];
    if(rank == 0) {
        for(int i = 0 ; i < DATA_SIZE * size ; i++){
            send[i] = i*i;
        }
    }
    MPI_Scatter(send, count, MPI_INT, recv, count, MPI_INT, root, MPI_COMM_WORLD);
    printf("Rank: %d\n", rank);
    for(int i = 0 ; i < count ; i++){
        printf("%d ", recv[i]);
    }
    printf("\n");
    MPI_Finalize();
    return 0;
}
```

Коллективные функции.

```
int MPI_Scatterv(void *sbuf, int *scounts,
int *displs, MPI_Datatype stype, void
*rbuf, int rcount, MPI_Datatype rtype, int
root, MPI_Comm comm)
```

Рассылка различного количества данных из массива `sbuf`. Начало рассылаемых порций задает массив `displs`.

Коллективные функции.

`scounts` – целочисленный массив, содержащий количество элементов, передаваемых каждому процессу (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).
`displs` – целочисленный массив, содержащий смещения относительно начала массива `sbuf` (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).

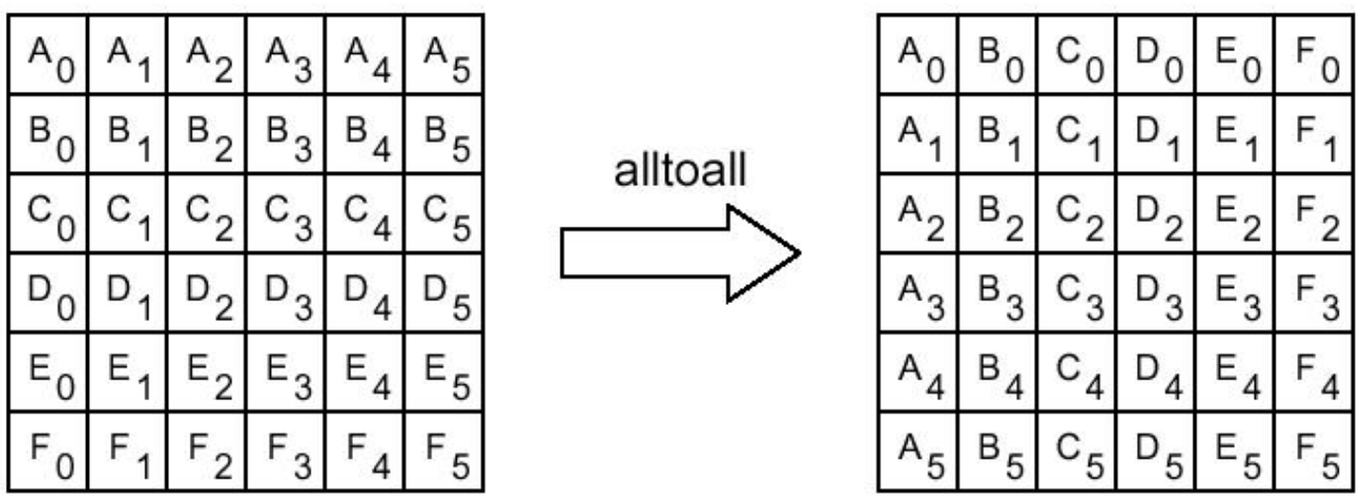
Коллективные функции.

```
int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(size != 3)
    {
        printf("This application is meant to be run with 3 processes.\n");
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    }
    int root_rank = 0;
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    switch(my_rank)
    {
        case 0:
        {
            int my_value;
            int buffer[7] = {100, 0, 101, 102, 0, 0, 103};
            int counts[3] = {1, 2, 1};
            int displacements[3] = {0, 2, 6};
            printf("Values in the buffer of root process:");
            for(int i = 0; i < 7; i++)
            {
                printf(" %d", buffer[i]);
            }
            printf("\n");
            MPI_Scatterv(buffer, counts, displacements, MPI_INT, &my_value, 1, MPI_INT, root_rank, MPI_COMM_WORLD);
            printf("Process %d received value %d.\n", my_rank, my_value);
            break;
        }
        case 1:
        {
            int my_values[2];
            MPI_Scatterv(NULL, NULL, NULL, MPI_INT, my_values, 2, MPI_INT, root_rank, MPI_COMM_WORLD);
            printf("Process %d received values %d and %d.\n", my_rank, my_values[0], my_values[1]);
            break;
        }
        case 2:
        {
            int my_value;
            MPI_Scatterv(NULL, NULL, NULL, MPI_INT, &my_value, 1, MPI_INT, root_rank, MPI_COMM_WORLD);
            printf("Process %d received value %d.\n", my_rank, my_value);
            break;
        }
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

Коллективные функции.

```
int MPI_Alltoall(void *sbuf, int
scount, MPI_Datatype stype, void
*rbuf, int rcount, MPI_Datatype
rtype, MPI_Comm comm)
```

Каждый процесс коммуникатора `comm` рассылает различные порции данных всем другим процессам. `j`-й блок массива `sbuf` процесса `i` попадает в `i`-й блок массива `rbuf` процесса `j`.



Коллективные функции.

Если для посылки и приема должен использоваться один буфер, то на месте аргумента `sbuf` всех процессов можно указать значение `MPI_IN_PLACE`. В этом случае аргументы `scount` и `stype` игнорируются, и предполагается, что порции исходных данных всех процессов уже расположены в соответствующих местах буферов приема `rbuf`.

Коллективные функции.

```
int MPI_Alltoallv(void* sbuf, int
*scounts, int *sdispls, MPI_Datatype
stype, void* rbuf, int *rcounts, int
*rdispls, MPI_Datatype rtype,
MPI_Comm comm)
```

Рассылка со всех процессов коммуникатора `comm` различного количества данных всем другим процессам. Размещение данных в буфере `sbuf` отсылающего процесса определяется массивом `sdispls`, а в буфере `rbuf` принимающего процесса – массивом `rdispls`.