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 моет быть разным. Все барьеры равнозначные.

Коллективные функции.

```
#define MAXPROC 128
#define NTIMES 10000
int main(int argc, char **argv)
{
   int rank, size, i, it;
   int ibuf[MAXPROC];
   double time_start, time_finish;
   MPI_Request req[2*MAXPROC]; MPI_Status statuses[MAXPROC];
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if(rank==0){
      for(i = 1; i<size; i++){
         MPI_Recv_init(&ibuf[i], 0, MPI_INT, i, 5, MPI_COMM_WORLD, &req[i]);
         MPI_Send_init(&rank, 0, MPI_INT, i, 6, MPI_COMM_WORLD, &req[size+i]);
      }
      time_start = MPI_Wtime();
      for(it = 0; it<NTIMES; it++){
         MPI_Startall(size-1, &req[1]);MPI_Waitall(size-1, &req[1], statuses);
         MPI_Startall(size-1, &req[size+1]);MPI_Waitall(size-1, &req[size+1], statuses);
      }
   }
   else{
      MPI_Recv_init(&ibuf[0], 0, MPI_INT, 0, 6, MPI_COMM_WORLD, &req[0]);
      MPI_Send_init(&rank, 0, MPI_INT, 0, 5, MPI_COMM_WORLD, &req[1]);
      time_start = MPI_Wtime();
      for(it = 0; it<NTIMES; it++){
         MPI_Start(&req[1]);
         MPI_Wait(&req[1], statuses);
         MPI_Start(&req[0]);
         MPI_Wait(&req[0], statuses);
      }
   }
   time_finish = MPI_Wtime()-time_start;
   printf("rank = %d model time = %lf\n", rank, time_finish/NTIMES);
   time_start = MPI_Wtime();
   for(it = 0; it<NTIMES; it++) MPI_Barrier(MPI_COMM_WORLD);
   time_finish = MPI_Wtime()-time_start;
   printf("rank = %d barrier time = %lf\n", rank, time_finish/NTIMES);
   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.

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.

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.

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.

```
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.

```
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.