Справочник по функциям MPI

Инициализация MPI

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

Инициализация параллельной части приложения. Реальная инициализация для каждого приложения выполняется не более одного раза, а если MPI уже был инициализирован, то никакие действия не выполняются и происходит немедленный возврат из подпрограммы. Все остальные MPI-процедуры могут быть вызваны только после вызова MPI_Init.

Возвращает: в случае успешного выполнения - MPI_SUCCESS, иначе - код ошибки. (То же самое возвращают и все остальные функции).

Сложный тип аргументов MPI_Init предусмотрен для того, чтобы передавать всем процессам аргументы main:

Пример вызова:

```
int main(int argc, char** argv)
{
      …
      MPI_Init(&argc, &argv);
      …
}
```

Закрытие MPI

int MPI_Finalize(void)

Завершение параллельной части приложения. Все последующие обращения к любым MPI-процедурам, в том числе к MPI_Init, запрещены. К моменту вызова MPI_Finalize любым процессом все действия по обмену сообщениями должны быть завершены.

Пример вызова:

```
int main(int argc, char** argv)
{
      …
      MPI_Finalize();
      …
}
```

Определение числа процессов

int MPI_Comm_size(MPI_Comm comm, int* size)

Определение общего числа параллельных процессов в группе comm (вместо comm во всех лабораторных работах использовать константу MPI_COMM_WORLD - группа "все процессы", связи в виде полного графа).

- comm - идентификатор группы

- выходной параметр size - размер группы. Здесь возвращается число процессов, которое пользователь задал при запуске программы.

Пример вызова:

```
…
int size;
MPI_Comm_size(MPI_COMM_WORLD, &size);
…
```

Определение номера процесса

int MPI_Comm_rank(MPI_comm comm, int* rank)

Определение номера процесса в группе comm. Возвращаемое значение (номер процесса, rank) лежит в диапазоне от 0 до size-1.

- comm - идентификатор группы
- выходной параметр rank - номер вызывающего процесса в группе comm

Пример вызова:

```
…
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
…
```

Аварийное завершение программы

int MPI_Abort(MPI_Comm comm, int errorcode)

Аварийное завершение работы всех процессов. Эта функция должна вызываться одновременно всеми процессами приложения.

- errorcode - код ошибки
- comm - идентификатор группы

Пример вызова:

```
…
MPI_Abort(MPI_COMM_WORLD,MPI_ERR_OTHER);
…
```

Передача сообщения

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

Посылка сообщения с меткой msgtag, состоящего из count элементов типа datatype, процессу с номером dest. Все элементы сообщения расположены подряд в буфере buf. Значение count может быть нулем. Тип передаваемых элементов datatype должен указываться с помощью предопределенных констант типа (для целых - MPI_INT).

Разрешается передавать сообщение самому себе. Метка должна быть одной и той же при приеме и передаче сообщения. Дальнейшее выполнение программы задерживается до тех пор, пока передача не завершится.

- buf - адрес начала буфера посылки сообщения
- count - число передаваемых элементов в сообщении
- datatype - тип передаваемых элементов
- dest - номер процесса-получателя
- msgtag - метка сообщения
- comm - идентификатор группы

Пример вызова:

```
…
#define N 10
…
int rank,buf[N];
…
MPI_Send(buf,N,MPI_INT,1,10,MPI_COMM_WORLD);
…
```

Прием сообщения

int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_comm comm, MPI_Status *status)

Прием сообщения с меткой msgtag от процесса source с блокировкой. Число элементов в принимаемом сообщении не должно превосходить значения count.

- выходной параметр buf - адрес начала буфера приема сообщения
- count - максимальное число элементов в принимаемом сообщении
- datatype - тип элементов принимаемого сообщения
- source - номер процесса-отправителя
- msgtag - метка принимаемого сообщения
- comm - идентификатор группы
- выходной параметр status - параметры принятого сообщения

Пример вызова:

```
…
#define N 10
…
int rank,buf[N];
MPI_Status status;
…
MPI_Recv(buf,N,MPI_INT,1,10,MPI_COMM_WORLD,&status);
…
```

Состояние полученных данных

Тип данных MPI_Status - это структура, содержащая следующие поля: MPI_SOURCE (источник), MPI_TAG (метка), MPI_ERROR (ошибка).

Пример вызова:

```
…
MPI_Status status;
int source;
…
MPI_Recv(buf,N,MPI_INT,1,10,MPI_COMM_WORLD,&status);
source = status.MPI_SOURCE;
…
```

Совмещенные прием/передача сообщений

int MPI_Sendrecv(void *sbuf, int scount, MPI_Datatype stype, int dest, int stag, void *rbuf, int rcount, MPI_Datatype rtype, int source, MPI_Dtatype rtag, MPI_Comm comm, MPI_Status *status)

Данная операция объединяет в едином запросе посылку и прием сообщений. Принимающий и отправляющий процессы могут являться одним и тем же процессом. Сообщение, отправленное операцией MPI_Sendrecv, может быть принято обычным образом, и точно также операция MPI_Sendrecv может принять сообщение, отправленное обычной операцией MPI_Send. Буфера приема и посылки обязательно должны быть различными.

- sbuf - адрес начала буфера посылки сообщения
- scount - число передаваемых элементов в сообщении
- stype - тип передаваемых элементов
- dest - номер процесса-получателя
- stag - метка посылаемого сообщения
- выходной параметр rbuf - адрес начала буфера приема сообщения
- rcount - число принимаемых элементов сообщения
- rtype - тип принимаемых элементов
- source - номер процесса-отправителя
- rtag - метка принимаемого сообщения
- comm - идентификатор группы
- выходной параметр status - параметры принятого сообщения

Пример вызова:

```
…
#define N 10
…
int rank,buf[N], buf1[N];
MPI_Status status;
…
MPI_Sendrecv(buf,N,MPI_INT,1,10,buf1,N,MPI_INT,0,10,MPI_COMM_WORLD,&status);
…
```

Проверка приемного буфера

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

Функция возвращает управление, когда в системном буфере процесса появляется сообщение с указанными параметрами. Если использовать аргументы-джокеры ("любой

источник", "любая метка"), как в примере ниже, то с помощью этой функции можно проверять наличие сообщений в системном буфере.

- source номер процесса-отправителя
- tag метка сообщения
- comm - идентификатор группы
- выходной параметр status - параметры принятого сообщения

Пример вызова:

```
…
MPI_Status status;
…
MPI_Probe(MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,&status);
…
```

Определение размера сообщения

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

Через параметр count возвращает длину сообщения. Обычно вызывается после MPI_Probe.

- status информация о сообщении
- datatype тип принимаемых элементов
- выходной параметр count - число элементов сообщения

Пример вызова:

```
…
MPI_Status status;
int count;
…
MPI_Get_count(&status,MPI_INT,&count);
…
```

Рассылка данных

int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int source, MPI_Comm comm)

Эта функция должна вызываться одновременно всеми процессами приложения. Рассылка сообщения от процесса source всем процессам, включая рассылающий процесс. При возврате из процедуры содержимое буфера buf процесса source будет скопировано в локальный буфер процесса. Значения параметров count, datatype и source должны быть одинаковыми у всех процессов.

- выходной параметр buf - адрес начала буфера посылки сообщения
- count - число передаваемых элементов в сообщении
- datatype - тип передаваемых элементов
- source - номер рассылающего процесса
- comm - идентификатор группы

Пример вызова:

```
…
#define N 10
…
int buf[N];
…
MPI_Bcast(buf,N,MPI_INT,0,MPI_COMM_WORLD);
…
```

Распределение данных

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

Части передающего буфера из задачи root распределяются по приемным буферам всех задач. Эта функция должна вызываться одновременно всеми процессами приложения.

- sbuf - адрес начала буфера посылки
- scount - число элементов в посылаемом сообщении; этот параметр должен быть равен размеру буфера, деленному на число процессов
- stype - тип элементов отсылаемого сообщения
- выходной параметр rbuf - адрес начала буфера сборки данных
- rcount - число элементов в принимаемом сообщении
- rtype - тип элементов принимаемого сообщения
- dest - номер процесса, на котором происходит сборка данных
- comm - идентификатор группы

Пример вызова:

```
…
#define N 10
#define PROCS 5
…
int buf[N],buf1[N/PROCS],size;
…
MPI_Scatter(buf,N/PROCS,MPI_INT, buf1,N/PROCS,MPI_INT,0,MPI_COMM_WORLD);
…
```

Сборка распределенных данных

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

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

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

Сборка данных со всех процессов в буфере rbuf процесса dest. Каждый процесс, включая dest, посылает содержимое своего буфера sbuf процессу dest. Собирающий процесс сохраняет данные в буфере rbuf, располагая их в порядке возрастания номеров процессов. Параметр rbuf имеет значение только на собирающем процессе и на остальных

игнорируется, значения параметров count, datatype и dest должны быть одинаковыми у всех процессов.

Для функции allgather и allgatherv все процессы собирают один вектор. Пусть у нас имеется вектор размером 8, который разделён между тремя процессами на 3 части размерами 3, 3 и 2 соответственно. Тогда массивы длин частей для векторного варианта функции примут следующий вид: recvcounts [3] = {3, 3, 2}, displs [3] = {0, 3, 6}.

- sbuf - адрес начала буфера посылки
- scount - число элементов в посылаемом сообщении; этот параметр должен быть равен размеру буфера, деленному на число процессов.
- stype - тип элементов отсылаемого сообщения
- выходной параметр rbuf - адрес начала буфера сборки данных
- rcount - число элементов в принимаемом сообщении (этот параметр должен быть равным scount)
- rtype - тип элементов принимаемого сообщения
- dest - номер процесса, на котором происходит сборка данных
- recvcounts массив, указывающий количество принимаемых элементов от процессов
- displs целочисленный массив смещений пакетов данных друг относительно друга
- comm - идентификатор группы

Пример вызова:

```
…
#define N 10
#define PROCS 5
…
int buf[N],buf1[N/PROCS],size;
…
MPI_Gather(buf1,N/PROCS,MPI_INT, buf,N/PROCS,MPI_INT,0,MPI_COMM_WORLD);
…
```

Синхронизация процессов

int MPI_Barrier(MPI_Comm comm)

Блокирует работу процессов, вызвавших данную процедуру, до тех пор, пока все оставшиеся процессы группы comm также не выполнят эту процедуру.

- comm - идентификатор группы

Пример вызова:

```
…
MPI_Barrier( MPI_COMM_WORLD);
…
```

Поэлементные операции

int MPI_Reduce (void *sbuf, void *rbuf, int count, MPI_Datatype stype; MPI_Op op, int dest, MPI_Comm comm)

int MPI_Allreduce (void *sbuf, void *rbuf, int count, MPI_Datatype stype; MPI_Op op, MPI_Comm comm)

Эта функция должна вызываться одновременно всеми процессами приложения. Все процессы подают на вход функции массивы buf одинаковой размерности count. Над этими массивами поэлементно выполняется операция op. Массив - результат помещается в процесс dest. Для операция allreduce массив размещается во всех процессах.

- sbuf - адрес начала буфера посылки
- rbuf - адрес начала буфера приема
- count - число элементов в посылаемом/принимаемом сообщении
- stype - тип элементов отсылаемого сообщения
- op - операция, выполняемая над элементами буфера
- dest - номер процесса, на котором происходит сборка данных
- comm - идентификатор группы

Виды поэлементных операций:

- MPI_MAX Выбор максимального элемента
- MPI_MIN Выбор минимального элемента
- MPI_SUM Суммирование
- MPI_PROD Вычисление произведения

Пример вызова:

```
…
#define N 10
int v[N],u[N];
…
MPI_Reduce(v,u,N,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
…
```