Технология программирования MPI (2)

Антонов Александр Сергеевич,

к.ф.-м.н., вед.н.с. лаборатории Параллельных информационных технологий НИВЦ МГУ

Суперкомпьютерная академия МГУ
Москва, 2022

MPI

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

MPI

Совмещённые приём и передача сообщений с блокировкой. Буферы передачи и приёма не должны пересекаться. Тупиковой ситуации не возникает!

Сообщение, отправленное операцией **`MPI_Sendrecv`**, может быть принято обычным образом, и операция **`MPI_Sendrecv`** может принять сообщение, отправленное обычной операцией.

MPI

```
int MPI_Sendrecv_replace(void
*buf, int count, MPI_Datatype
datatype, int dest, int stag,
int source, int rtag, MPI_comm
comm, MPI_Status *status)
```

Совмещённые приём и передача сообщений с блокировкой через общий буфер **`buf`**.

Принимаемое сообщение не должно превышать по размеру отправляемое, а данные должны быть одного типа.

MPI

Обмен по кольцевой топологии при помощи процедуры **MPI_Sendrecv**:

```
prev = rank - 1;
next = rank + 1;
if (rank == 0) prev = size - 1;
if (rank == (size - 1)) next = 0;
MPI_Sendrecv(&sbuf[0], 1, MPI_FLOAT, prev,
tag2, &rbuf[0], 1, MPI_FLOAT, next, tag2,
MPI_COMM_WORLD, &status1);
MPI_Sendrecv(&sbuf[1], 1, MPI_FLOAT, next,
tag1, &rbuf[1], 1, MPI_FLOAT, prev, tag1,
MPI_COMM_WORLD, &status2);
```

MPI

Передача и приём сообщений без блокировки

MPI

В MPI предусмотрен набор процедур для осуществления асинхронной передачи данных. В отличие от блокирующих процедур, возврат из процедур данной группы происходит сразу после вызова без какой-либо остановки работы процессов. После этого на фоне дальнейшего выполнения программы одновременно может происходить и завершение асинхронно запущенной операции.

MPI

Для получения информации о завершении асинхронного обмена требуется вызов дополнительной процедуры, которая проверяет, завершилась ли операция, или дожидается ее завершения. Только после этого можно использовать буфер посылки для других целей без опасения испортить отправляемое сообщение.

MPI

```
int MPI_Isend(void *buf, int
count, MPI_Datatype datatype,
int dest, int msgtag, MPI_Comm
comm, MPI_Request *request)
```

Неблокирующая посылка сообщения. Возврат из функции происходит сразу после инициализации передачи. Переменная **`request`** идентифицирует пересылку.

MPI

Определить тот момент времени, когда можно повторно использовать буфер **`buf`** без опасения испортить передаваемое сообщение, можно с помощью возвращаемого параметра **`request`** и процедур семейств **`MPI_Wait`** и **`MPI_Test`**.

MPI

Модификации функции **`MPI_Isend`**:

- **`MPI_Ibsend`** — передача сообщения с буферизацией;
- **`MPI_Issend`** — передача сообщения с синхронизацией;
- **`MPI_Irsend`** — передача сообщения по готовности.

MPI

```
int MPI_Irecv(void *buf, int
count, MPI_Datatype datatype,
int source, int msgtag, MPI_Comm
comm, MPI_Request *request)
```

Неблокирующий приём сообщения. Возврат из функции происходит сразу после инициализации передачи. Переменная **`request`** идентифицирует пересылку.

MPI

Сообщение, отправленное любой из процедур **`MPI_Send`**, **`MPI_Isend`** и любой из трёх их модификаций, может быть принято любой из процедур **`MPI_Recv`** и **`MPI_Irecv`**.

До завершения неблокирующей операции не следует записывать в используемый массив данных!

MPI

```
int MPI_Iprobe(int source, int
msgtag, MPI_Comm comm, int
*flag, MPI_Status *status)
```

Получение информации о структуре ожидаемого сообщения без блокировки. В аргументе **`flag`** возвращается значение **`1`**, если сообщение с подходящими атрибутами уже может быть принято, и значение **`0`**, если сообщения с указанными атрибутами еще нет.

MPI

```
int MPI_Wait(MPI_Request
*request, MPI_Status *status)
```

Ожидание завершения асинхронной операции, ассоциированной с идентификатором **`request.`** Для неблокирующего приёма определяется параметр **`status. request`** устанавливается в значение **`MPI_REQUEST_NULL.`**

MPI

```
int MPI_Waitall(int count,
MPI_Request *requests,
MPI_Status *statuses)
```

Ожидание завершения **`count`** асинхронных операций, ассоциированных с идентификаторами **`requests`**. Для неблокирующих приёмов определяются параметры в массиве **`statuses`**.

MPI

Если во время одной или нескольких операций обмена возникли ошибки, то поля ошибки в элементах массива **`statuses`** будут установлены в соответствующие этим ошибкам значения. После выполнения процедуры соответствующие элементы параметра **`requests`** устанавливаются в значение **`MPI_REQUEST_NULL`**.

MPI

Обмен по кольцевой топологии при помощи неблокирующих операций:

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
   int rank, size, prev, next;
   int buf[2];
   MPI_Request reqs[4];
   MPI_Status stats[4];
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

MPI

```
   prev = rank - 1;
   next = rank + 1;
   if(rank==0) prev = size - 1;
   if(rank==size - 1) next = 0;
   MPI_Irecv(&buf[0], 1, MPI_INT, prev, 5, MPI_COMM_WORLD,
&reqs[0]);
   MPI_Irecv(&buf[1], 1, MPI_INT, next, 6, MPI_COMM_WORLD,
&reqs[1]);
   MPI_Isend(&rank, 1, MPI_INT, prev, 6, MPI_COMM_WORLD,
&reqs[2]);
   MPI_Isend(&rank, 1, MPI_INT, next, 5, MPI_COMM_WORLD,
&reqs[3]);
   MPI_Waitall(4, reqs, stats);
   printf("process %d prev = %d next=%d\n", rank, buf[0],
buf[1]);
   MPI_Finalize();
}
```

MPI

```
int MPI_Waitany(int count,
MPI_Request *requests, int
*index, MPI_Status *status)
```

Ожидание завершения одной из **count** асинхронных операций, ассоциированных с идентификаторами **requests**. Для неблокирующего приёма определяется параметр **status**.

MPI

Если к моменту вызова завершились несколько из ожидаемых операций, то случайным образом будет выбрана одна из них.

Параметр **`index`** содержит номер элемента в массиве **`requests`**, содержащего идентификатор завершённой операции. В языке Си массив индексируется с 0.

Соответствующий элемент **`requests`** устанавливается в **`MPI_REQUEST_NULL`**.

MPI

```
int MPI_Waitsome(int incount,
MPI_Request *requests, int
*outcount, int *indexes,
MPI_Status *statuses)
```

Ожидание завершения хотя бы одной из **`incount`** асинхронных операций, ассоциированных с идентификаторами **`requests`**.

MPI

`outcount` содержит число завершённых операций, а первые **`outcount`** элементов **`indexes`** содержат номера элементов **`requests`** с их идентификаторами.

Первые **`outcount`** элементов массива **`statuses`** содержат параметры завершённых операций (для неблокирующих приёмов).

Соответствующий элемент **`requests`** устанавливается в **`MPI_REQUEST_NULL`**.

MPI

Схема «мастер – рабочие» :

```
#include <stdio.h>
#include "mpi.h"
#define N 1000
#define MAXPROC 128
void slave(double *a, int n){/* обработка локальной части массива a */}
void master(double *a, int n){/* обработка массива a */}
int main(int argc, char **argv)
{
   int rank, size, num, k, i, indices[MAXPROC], source;
   MPI_Request req[MAXPROC];
   MPI_Status statuses[MAXPROC];
   double a[N][MAXPROC];
```

MPI

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if(rank!=0)
    while(1){
        slave((double*)a, N);
        MPI_Send(a, N, MPI_DOUBLE, 0, 5, MPI_COMM_WORLD);
    }
else{
    for(i = 0; i<size-1; i++)
        MPI_Irecv(&a[0][i], N, MPI_DOUBLE, i, 5,
MPI_COMM_WORLD, &req[i]);
    while(1){
        MPI_Waitsome(size-1, req, &num, indices,
statuses);
```

MPI

```
            for (i = 0; i< num; i++){
                source = statuses[i].MPI_SOURCE;
                master(&a[0][source], N);
                MPI_Irecv(&a[0][source], N, MPI_DOUBLE,
source, 5, MPI_COMM_WORLD, &req[source]);
            }
        }
    }
  MPI_Finalize();
}
```

MPI

```
int MPI_Test(MPI_Request
*request, int *flag, MPI_Status
*status)
```

Проверка завершённости асинхронной операции, ассоциированной с идентификатором **`request.`** В параметре **`flag`** возвращается значение **`1`**, если операция завершена, и значение **`0`** – в противном случае..

MPI

Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра **`status`**. После выполнения процедуры соответствующий элемент параметра **`request`** устанавливается в значение **`MPI_REQUEST_NULL`**.

MPI

```
int MPI_Testall(int count,
MPI_Request *requests, int
*flag, MPI_Status *statuses)
```

Проверка завершённости **`count`** асинхронных операций, ассоциированных с идентификаторами **`requests`**. В параметре **`flag`** возвращает **`1`**, если все указанные операции завершены. Параметры принимаемых сообщений будут в массиве **`statuses`**.

MPI

Если какая-либо из операций не завершилась, то возвращается **0**, и определенность элементов массива **statuses** не гарантируется. После выполнения процедуры соответствующие элементы параметра **requests** устанавливаются в значение **MPI_REQUEST_NULL**.

MPI

```
int MPI_Testany(int count,
MPI_Request *requests, int
*index, int *flag, MPI_Status
*status)
```

В параметре **flag** возвращается значение **1**, если хотя бы одна из операций асинхронного обмена завершена, при этом **index** содержит номер соответствующего элемента в массиве **requests**, а **status** – параметры принимаемого сообщения.

MPI

Если ни одна из операций не завершилась, то в параметре **`flag`** будет возвращено значение **`0`**. Если к моменту вызова завершились несколько ожидаемых операций, то случайным образом будет выбрана одна из них. После выполнения процедуры соответствующий элемент параметра **`requests`** устанавливается в значение **`MPI_REQUEST_NULL`**.

MPI

```
int MPI_Testsome(int incount,
MPI_Request *requests, int
*outcount, int *indexes,
MPI_Status *statuses)
```

Аналог **MPI_Waitsome,** но возврат происходит немедленно. Если ни одна из операций не завершилась, то значение **outcount** будет равно нулю.

MPI

Отложенные запросы на взаимодействие

MPI

Часто в программе приходится многократно выполнять обмены с одинаковыми параметрами (например, в цикле). В этом случае можно один раз инициализировать операцию обмена и потом многократно ее запускать, не тратя на каждой итерации дополнительного времени на инициализацию и заведение соответствующих внутренних структур данных.

MPI

Кроме того, несколько запросов на прием и/или передачу могут объединяться вместе для того, чтобы далее их можно было бы запустить одной командой (впрочем, это совсем не обязательно хорошо, поскольку может привести к перегрузке коммуникационной сети).

MPI

```
int MPI_Send_init(void *buf, int
count, MPI_Datatype datatype,
int dest, int msgtag, MPI_Comm
comm, MPI_Request *request)
```

Формирование отложенного запроса на посылку сообщения. Сама операция пересылки не начинается!

MPI

Модификации функции **`MPI_Send_init`**:

- **`MPI_Bsend_init`** — формирование запроса на передачу сообщения с буферизацией;
- **`MPI_Ssend_init`** — формирование запроса на передачу сообщения с синхронизацией;
- **`MPI_Rsend_init`** — формирование запроса на передачу сообщения по готовности.

MPI

```
int MPI_Recv_init(void *buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Request *request)
```

Формирование отложенного запроса на приём сообщения. Сама операция приёма не начинается!

MPI

```
int MPI_Start(MPI_Request *request)
```

Инициализация отложенного запроса на выполнение операции обмена, соответствующей значению параметра **`request.`** Операция запускается как неблокирующая.

MPI

```
int MPI_Startall(int count,
MPI_Request *requests)
```

Инициализация **count** отложенных запросов на выполнение операций обмена, соответствующих значениям первых **count** элементов массива **requests.** Операции запускаются как неблокирующие.

MPI

Сообщение, отправленное при помощи отложенного запроса, может быть принято любой из процедур **`MPI_Recv`** и **`MPI_Irecv`**, и наоборот.

По завершении отложенного запроса значение параметра **`request (requests)`** сохраняется и может использоваться в дальнейшем!

MPI

```
int MPI_Request_free(MPI_Request *request)
```

Удаляет структуры данных, связанные с параметром **`request. request`** устанавливается в значение **`MPI_REQUEST_NULL`**. Если операция, связанная с этим запросом, уже выполняется, то она завершится.

MPI

Схема итерационного метода с обменом по кольцевой топологии при помощи отложенных запросов:

```
prev = rank - 1;
next = rank + 1;
if (rank == 0) prev = size - 1;
if (rank == (size - 1)) next = 0;
MPI_Recv_init(&rbuf[0], 1, MPI_FLOAT, prev, tag1,
MPI_COMM_WORLD, &reqs[0]);
MPI_Recv_init(&rbuf[1], 1, MPI_FLOAT, next, tag2,
MPI_COMM_WORLD, &reqs[1]);
MPI_Send_init(&sbuf[0], 1, MPI_FLOAT, prev, tag2,
MPI_COMM_WORLD, &reqs[2]);
MPI_Send_init(&sbuf[1], 1, MPI_FLOAT, next, tag1,
MPI_COMM_WORLD, &reqs[3]);
```

MPI

```
for(…){
 sbuf[0]=…;
 sbuf[1]=…;
 MPI_Startall(4, reqs);
 ...
 MPI_Waitall(4, reqs, stats);
 ...
}
MPI_Request_free(&reqs[0]);
MPI_Request_free(&reqs[1]);
MPI_Request_free(&reqs[2]);
MPI_Request_free(&reqs[3]);
```

MPI

```
int MPI_Request_get_status(
MPI_Request request, int *flag,
MPI_Status *status)
```

Позволяет получить информацию об асинхронной операции, ассоциированной с идентификатором **`request`**, без освобождения соответствующих структур данных. В параметре **`flag`** возвращается значение **`1`**, если операция завершена, и значение **`0`** – в противном случае.

MPI

Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра **`status`**. Однако после выполнения этой процедуры параметр **`request`** остается неизменным.

MPI

```
int MPI_Cancel(MPI_Request *request)
```

Запускает процесс отмены асинхронной операции, ассоциированной с идентификатором **`request`**. Возврат не означает, что соответствующая операция отменена. После вызова необходимо вызвать одну из функций **`MPI_Request_free`**, **`MPI_Wait`**, **`MPI_Test`** или их производных.

MPI

Успешная отмена неблокирующей передачи сообщения с буферизацией приводит к освобождению буфера, в котором сообщение ожидает отправки. Возможна только либо успешная отмена асинхронной операции, либо ее полное выполнение. Информация об успешной отмене асинхронной операции заносится в объект **`status`**, соответствующий завершаемой операции.

MPI

```
int MPI_Test_cancelled(
MPI_Status *status, int *flag)
```

Возвращает в параметре **flag** значение **1**, если операция, ассоциированная с операцией, заданной **status**, была успешно отменена, и **0** – в противном случае. Если операция неблокирующего приема могла быть отменена, то прежде чем обращаться к полям **status**, нужно проверить отмену при помощи **MPI_Test_cancelled**.

MPI

Задание 5: Напишите программу на MPI, реализующую транспонирование квадратной матрицы, распределенной между процессорами по строкам (столбцам), с использованием неблокирующих операций. Исследуйте зависимость времени выполнения программы от размера матриц и количества процессов.