Параллельные алгоритмы: MPI. Взаимодействия типа точка-точка

Н. И. Хохлов

МФТИ, Долгопрудный

21 февраля 2018 г.

Взаимодействия точка-точка. Особенности

- Всего учавствуют *два* и только два процесса.
- Явный процесс взаимодействия. Один процесс всегда принимает данные, другой отсылает.
- Существует несколько типов взаимодействия, операции приема/отсылки разных типом могут комбинироваться.
- Процессы обмениваются данными, только если состоят в одном коммуникаторе (в рамках одного контекста).

Общий интерфейс

Функции отсылки и приема построены по единому интерфейсу.

Example

send(address, count, datatype, destination, tag, comm)

и

secv(address, maxcount, datatype, source, tag, comm, status)

Типы пересылок

- Четыре типа операций отсылки и одна операция приема.
- Завершение операции гарантирует безопасность дальнейшего использования буфера отсылки (изменения никак не скажутся на стороне получателе).
- Все операции существуют в блокирующем и асинхронном виде.
- Несколько процедур взаимодействия могут быть объдинены для ускорения отсылки (persistent communication).

Типы пересылок

Тип взаимодействия	**Условие завершения**
Синхронная отсылка (synchronous)	Завершается только после начала приема
Буферезированная отсылка (buffered)	Завершается всегда, не гарантирует прием
Обычная отсылка (standard)	Работает как синхронная или буферезированная
Отсылка по готовности (ready)	Завершается всегда, не гарантирует прием
Прием (Receive)	Завершается когда сообщение доставлено

Реализация в MPI

Тип взаимодействия	**Блокирующая операция**
Синхронная отсылка	MPI_Ssend
Буферезированная отсылка	MPI_Bsend
Обычная отсылка	MPI_Send
Отсылка по готовности	MPI_Rsend
Прием	MPI_Recv

MPI_Send

Стандартная отсылка

- Завершается как только буфер становится безопасен для дальнейшего использования.
- Не гарантирует доставку и даже того, что начался прием (сообщение может находится в системном буфере).
- Может быть реализована через синхронную или буферезированную отсылку (или их комбинацию).
- В зависимости от реализации поведение может отличаться и при написании приложений следует рассматривать работу функции как синхронную.
- Множество операций отсылок без приемов может загружать сеть.

MPI_Send. Синтаксис

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

- buf – адрес начала расположения пересылаемых данных;
- count – число пересылаемых элементов;
- type – MPI-тип посылаемых элементов;
- dest – номер процесса-получателя в группе, связанной с коммуникатором comm;
- tag – идентификатор сообщения;
- comm – коммуникатор.

Типы данных

Типы данных можно создавать для различных типов языка.
Существуют глобальные типы для встроенных типов языка.

Тип C	**Тип MPI**
int	MPI_INT
float	MPI_FLOAT
char	MPI_CHAR
double	MPI_DOUBLE
long	MPI_LONG
long double	MPI_LONG_DOUBLE

Примеры

Передача числа

```
int a;
...
MPI_Send(&a, 1, MPI_INT, rank, tag, MPI_COMM_WORLD);
```

Передача статического массива

```
int a[5];
...
MPI_Send(a, 5, MPI_INT, rank, tag, MPI_COMM_WORLD);
```

или

```
MPI_Send(&a[0], 5, MPI_INT, rank, tag, MPI_COMM_WORLD);
```

Примеры

Передача динамического массива

```
int *a = (int*)malloc(sizeof(int) * 5);
...
MPI_Send(a, 5, MPI_INT, rank, tag, MPI_COMM_WORLD);
```

Передача части статического или динамического массива

```
int a[5]; (или int *a = (int*)malloc(sizeof(int) * 5);)
...
MPI_Send(a+1, 2, MPI_INT, rank, tag, MPI_COMM_WORLD);
```

Будут отправлены элементы с 1-го по 3-й.

MPI_Recv

Базовая функция приема

- Завершение гарантирует, что все данные приняты.
- Может принять данные от любой функции отсылки.
- Может принять меньше данных чем указано, но не может больше.

MPI_Recv. Синтаксис

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

- buf – адрес начала буффера приема;
- count – максимальный размер буффера;
- type – MPI-тип принимаемых данных;
- source – номер процесса-отправителя в группе, связанной с коммуникатором comm;
- tag – идентификатор сообщения;
- comm – коммуникатор;
- status – статус принятого сообщения.

MPI_Status

Специальная структура, хранящая статус принятого сообщения. В ней описываются размер принятого сообщения, номер процесса отправителя и тег сообщения.
Поля структуры:

- MPI_SOURCE – номер процесса отправителя;
- MPI_TAG – тег принятого сообщения;

Размер реально принятого сообщения можно узнать через функцию MPI_Get_count.

MPI_Get_count. Синтаксис

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

- status – статус сообщения, размер которого требуется узнать;
- datatype – в каких типах требуется размер;
- count – размер сообщения в типах datatype. Если размер сообщения не кратен типам datatype, то вернется MPI_UNDEFINED.

MPI_Recv. Заглушки

- MPI_ANY_SOURCE – может быть указана вместо аргумента source, тогда сообщение будет принято от любого процессора в коммуникаторе comm;
- MPI_ANY_TAG – может быть указана в качестве аргумента tag, будет принято сообщение с любым тегом;
- MPI_STATUS_IGNORE – может быть указано вместо аргумента status, тогда статус сообщения будет проигнорирован.

Возможны любые комбинации используемых констант.

Примеры

Прием числа

```
int a;
...
MPI_Recv(&a, 1, MPI_INT, rank, tag,
         MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Прием массива

```
int a[5]; (или динамический)
...
MPI_Recv(a, 5, MPI_INT, rank, tag,
         MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

или

```
MPI_Recv(&a[0], 5, MPI_INT, rank, tag,
         MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Правила пересылок

- Сообщения от одного процесса не обгоняют друг друга.
- Сообщения от различных процессов могут приходить в произвольном порядке.

Пример пересылок

```
int rank, count;
char buf[100];
MPI_Status status;
MPI_Comm_rank(comm, &rank);
if (rank == 0) {
    strcpy(buf, "Hello from 0");
    MPI_Send(buf, strlen(buf) + 1, MPI_CHAR, 1, 99, comm);
} else if (rank == 1) {
    MPI_Recv(buf, 100, MPI_CHAR, MPI_ANY_SOURCE,
             MPI_ANY_TAG, comm, &status);
    MPI_Get_count(&status, MPI_CHAR, &count);
    printf("Message '%s', from %d, tag %d, size %d\n
           buf,
           status.MPI_SOURCE,
           status.MPI_TAG,
           count);
}
```

Практические задания

- Реализовать пересылку от одного процесса к другому.
- Проверить что будет при несовпадении тегов, номеров процессов.
- Проверить работу заглушек MPI_ANY_TAG, MPI_ANY_SOURCE.

Спасибо за внимание! Вопросы?