Многопоточные вычисления на основе технологий MPI и OpenMP: Взаимодействия типа точка-точка

Н. И. Хохлов

МФТИ, Долгопрудный

5 октября 2016 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**	**Асинхронная операция**
Синхронная отсылка	MPI_Ssend	MPI_Issend
Буферезированная отсылка	MPI_Bsend	MPI_Ibsend
Обычная отсылка	MPI_Send	MPI_Isend
Отсылка по готовности	MPI_Rsend	MPI_Irsend
Прием	MPI_Recv	MPI_Irecv

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
*	MPI_BYTE
*	MPI_PACKED

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_Ssend

Синхронная отсылка

- Завершение гарантирует начало приема.
- Не гарантирует, что операция приема завершена.
- Синтаксис аналогичес MPI_Send.
- Несет меньшуб нагрузку на сеть, чем MPI_Send, но в ряде случаев может работать медленнее.
- Необходимо следить за возможными блокировками.

MPI_Bsend

Буфферезированная отсылка

- Завершение гарантирует безопасность буфера отсылаемого сообщения.
- Не гарантирует прием.
- Синтаксис аналогичес MPI_Send.
- Пользователь должен явно управлять буффером для сообщений.

Работа с буфером сообщений

Подключение буффера
int MPI_Buffer_attach (void* buffer, int size);

- buffer – адрес начала буффера, должен быть выделен пользователем заранее;
- size – размер буффера в байтах.

Отключение буфера
int MPI_Buffer_detach(void* buffer_addr, int* size);

- buffer_addr – указатель на буффер, который используется. Фактически это void**;
- size – размер буффера в байтах.

Если пользователь не подключил буффер, то MPI считает что его размер нулевой.

Работа с буфером сообщений

Что будет, если пользователь вызвал MPI_Buffer_detach, а сообщения еще не доставлены?

Функция будет завершена, когда завершатся все операции буферезированных пересылок.

Работа с буфером сообщений

Как узнать размер буффера, который надо выделить?

Необходимо выделить столько места, чтобы туда поместились все отсылаемые сообщения + некое дополнительное пространство. Верхний предел дополнительного пространства описывается константой MPI_BSEND_OVERHEAD.

Для определения размера сообщения, занимаемого в буфере существует функция
int MPI_Pack_size(int incount, MPI_Datatype datatype, MPI_Comm comm, int *size);

- incount – число элементов в сообщении;
- datatype – тип пересылаемых данных;
- comm – коммуникатор, в рамках которого идет обмен;
- size – размер буффера в байтах.

Пример

Вычисление размера буффера

```
MPI_Pack_size(20, type1, comm, &s1);
MPI_Pack_size(40, type2, comm, &s2);
size = s1 + s2 + 2 * MPI_BSEND_OVERHEAD;
...
buf = allocate size bytes
...
MPI_Buffer_attach(buffer, size);
...
MPI_Bsend(..., count=20, datatype=type1,  ...);
...
MPI_Bsend(..., count=40, datatype=type2, ...);
```

Работа с буфером сообщений

Для чего MPI_Buffer_detach возвращает размер и указатель на буффер сообщений?

Одновременно в MPI можно работать только с одним буффером. Если какая-то сторонняя библиотека использует буффер, она может отключить текущий буффер, произвести обмен и подключить старый буффер.

Пример

Отключение и подключение буффера

```
int size, mysize, idummy;
void *ptr, *myptr, *dummy;
...
MPI_Buffer_detach(&ptr, &size);
MPI_Buffer_attach(myptr, mysize);
...
... library code ...
...
MPI_Buffer_detach(&dummy, &idummy);
MPI_Buffer_attach(ptr, size);
```

MPI_Rsend

Отсылка по готовности

- Завершение гарантирует безопасность буфера отсылаемого сообщения.
- Не гарантирует прием.
- Синтаксис аналогичес MPI_Send.
- Корректность работы функции гарантируется только если до ее вызова была вызвана сообветствующая функция приема.
- В ряде случаев может работать быстрее из-за отсутствия проверок.

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.

Правила сообветствия типов

- Типы данных в соответствующих операциях приема и отсылки должны совпадать.
- Исключение составляет MPI_PACKED (упакованные данные).
- Для отправки произвольных данных можно использовать MPI_BYTE, тогда данные будут пересылаться в неизменном двоичном виде.

Правила сообветствия типов

Корректная работа

```
int *a, *b;
...
MPI_Comm_rank(comm, &rank);
if (rank == 0) {
    MPI_Send(a, 5, MPI_INT, 1, tag, comm);
} else {
    MPI_Recv(b, 10, MPI_INT, 0, tag, comm, &st);
}
```

Типы a, b должны быть int, а размер не менее 5.

Правила сообветствия типов

Некорректная работа

```
int *a, *b;
...
MPI_Comm_rank(comm, &rank);
if (rank == 0) {
    MPI_Send(a, 5, MPI_INT, 1, tag, comm);
} else {
    MPI_Recv(b, sizeof(int) * 10, MPI_BYTE, 0, tag, comm,
}
```

Типы a, b не совпадают.

Правила сообветствия типов

Корректная работа

```
int *a, *b;
...
MPI_Comm_rank(comm, &rank);
if (rank == 0) {
    MPI_Send(a, sizeof(int) * 5, MPI_BYTE, 1, tag, comm);
} else {
    MPI_Recv(b, sizeof(int) * 10, MPI_BYTE, 0, tag, comm,
}
```

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);
}
```

Асинхронные (неблокирующие) взаимодействия

- Существуют асинхронные аналоги всех базовых функций взаимодействия.
- Аргументы у функций аналогичные, за исключением последнего аргумента – идентификатора неблокирующего взаимодействия request.

Тип данных для идентификатора неблокирубщего взаимодействия MPI_Request.

Синтаксис

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

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

Аналогочно для других типов взаимодействия.

Проверка завершения

Завершение неблокирующего взаимодействия аналогично завершению его блокирубщего аналога. Для проверки завершения операции существую процедуры MPI_Test и MPI_Wait.

Процедура MPI_Wait дожидается завершения неблокирубщего взаимодействия. Последовательный вызов процедуры неблокирубщего взаимодействия и MPI_Wait аналогично соответствующему блокирубщему вызову.

Процедура MPI_Test проверяет статус завершения неблокирующего взаимодействия (завершено или нет).

Синтаксис

int MPI_Wait(MPI_Request *request, MPI_Status *status)

- request – хендлер неблокирубщего взаимодействия;
- status – статус принятого сообщения (для операций отсылки), можно использовать MPI_STATUS_IGNORE.

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

- request – хендлер неблокирубщего взаимодействия;
- flag – флан завершения, 1 – завершено, 0 – не завершено;
- status – статус принятого сообщения (для операций отсылки), можно использовать MPI_STATUS_IGNORE.

Проверка завершения нескольких операций

Существуют аналоги MPI_Test и MPI_Wait для нескольких операций.

Проверка	**Wait** (блокировка и ожидание)	**Test** (проверка)
Как минимум одна	MPI_Waitany	MPI_Testany
Все	MPI_Waitall	MPI_Testall
Какие либо	MPI_Waitsome	MPI_Testsome

MPI_Waitany. Синтаксис

int MPI_Waitany(int count, MPI_Request array_of_requests[], int *index, MPI_Status *status);

- count – размер массива array_of_requests;
- array_of_requests – массив с хендлерами неблокирубщих взаимодействий;
- index – индекс в массиве взаимодействия, которое завершено;
- status – статус завершенного взаимодействия, можно использовать MPI_STATUS_IGNORE..

Дожидается выполнения только одного взаимодействия.

MPI_Waitall. Синтаксис

int MPI_Waitall(int count, MPI_Request array_of_requests[], MPI_Status array_of_statuses[]);

- count – размер массива array_of_requests;
- array_of_requests – массив с хендлерами неблокирубщих взаимодействий;
- array_of_statuses – массив статусов неблокирубщих взаимодействий, можно использовать MPI_STATUSES_IGNORE.

Дожидается выполнения всех взаимодействий.

MPI_Waitsome. Синтаксис

int MPI_Waitsome(int incount, MPI_Request array_of_requests[], int *outcount, int array_of_indices[], MPI_Status array_of_statuses[]);

- incount – размер массива array_of_requests;
- array_of_requests – массив с хендлерами неблокирубщих взаимодействий;
- outcount – число завершенных взаимодействий;
- array_of_indices – массив с номерами неблокирубщих взаимодействий;
- array_of_statuses – массив статусов завершенных неблокирубщих взаимодействий, можно использовать MPI_STATUSES_IGNORE.

Аналогично MPI_Waitany, но может вернуть больше одного неблокирубщего взаимодействия.

MPI_Testany. Синтаксис

int MPI_Testany(int count, MPI_Request array_of_requests[], int *index, int *flag, MPI_Status *status);

- count – размер массива array_of_requests;
- array_of_requests – массив с хендлерами неблокирубщих взаимодействий;
- index – индекс в массиве взаимодействия, которое завершено;
- flag – флан завершения, 1 – завершено, 0 – не завершено;
- status – статус завершенного взаимодействия, можно использовать MPI_STATUS_IGNORE..

Выполняет проверку только одного взаимодействия.

MPI_Testall. Синтаксис

int MPI_Testall(int count, MPI_Request array_of_requests[], int *flag, MPI_Status array_of_statuses[]);

- count – размер массива array_of_requests;
- array_of_requests – массив с хендлерами неблокирубщих взаимодействий;
- flag – флан завершения всех взаимодействий, 1 – завершено, 0 – не завершено;
- array_of_statuses – массив статусов неблокирубщих взаимодействий, можно использовать MPI_STATUSES_IGNORE.

Проверяет выполнения всех взаимодействий.

MPI_Testsome. Синтаксис

int MPI_Testsome(int incount, MPI_Request array_of_requests[], int *outcount, int array_of_indices[], MPI_Status array_of_statuses[]);

- incount – размер массива array_of_requests;
- array_of_requests – массив с хендлерами неблокирубщих взаимодействий;
- outcount – число завершенных взаимодействий;
- array_of_indices – массив с номерами неблокирубщих взаимодействий;
- array_of_statuses – массив статусов завершенных неблокирубщих взаимодействий, можно использовать MPI_STATUSES_IGNORE.

Аналогично MPI_Waitany, за исключением того, что результат выдает сразу, и если нет завершенных взаимодействий, то outcount будет 0.

Задание 1

Задание к следующему семинару. Прислать на почту mipt.courses@gmail.com в формате pdf и исходный код.

Измерение латентности передачи сообщений (1 балл)

Построить график времени передачи сообщения от его размера (подобрать масштаб для лучшего отображения). Размеры сообщения от 1Б до 1МБ. Построить для типов сообщений MPI_Send, MPI_Ssens, MPI_Bsend.

Измерение скорости передачи сообщений (1 балл)

Построить график скорости (ед.данных в ед.времени) передачи сообщения от его размера (подобрать масштаб для лучшего отображения). Размеры сообщения от 1Б до 1МБ. Построить для типов сообщений MPI_Send, MPI_Ssens, MPI_Bsend.

Спасибо за внимание! Вопросы?