Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс

Теория и практика параллельных вычислений

Лекция 8

Методы разработки параллельных программ при использования интерфейса передачи сообщений MPI–2

Гергель В.П.

Содержание

Конструирование производных типов данных в MPI: постановка проблемы

- Эффективность параллельность вычислений в распределенной памятью достижима только при минимизации операций пересылки данных
- Параметр MPI_Datatype в функциях передачи данных может иметь только определенный в MPI тип

? Если нужно передать данные, не описываемые в целом базовым типом MPI, то необходима последовательность операций передач данных?

→ *В MPI возможным является конструирование производных (пользовательских) типов* !

Базовые типы данных MPI

MPI_Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Понятие производного типа данных…

Под **производным типом данных** в MPI понимается последовательность данных, описываемая набором

$$\{(type_0, disp_0),\ldots,(type_{n-1}, disp_{n-1})\}$$

где

- **$type_i$** – есть базовый тип ***i*** элемента типа;
- **$disp_i$** – есть смещение в байтах для ***i*** элемента типа от начала значения типа.

Подобный набор описателей составных элементов называется в MPI *картой типа* (*datamap*).

Список описателей, состоящий только из указания типов элементов, называется *сигнатурой типа* (*type signature*)

$$\{(type_0,\ldots,type_{n-1}\}$$

Понятие производного типа данных

Пример:

```
double a; /* адрес 24 */
double b; /* адрес 40 */
int    n; /* адрес 48 */
```

Карта типа

```
{(MPI_DOUBLE,0),
 (MPI_DOUBLE,16),
 (MPI_INT,24)
}
```

Общий способ конструирования производных типов данных…

```
int MPI_Type_struct(int count, int blocklens[],
 MPI_Aint offsets[], MPI_Datatype types[],
 MPI_Datatype *newtype);
```

где

- **count** – количество элементов в типе;
- **blocklens[]** – количества значений в элементах типа;
- **offsets[]** – смещения элементов от начала значения типа;
- **types[]** – типы значений элементов типа;
- **newtype** – имя переменной для ссылки на тип.

После использования переменная-дескриптор типа должна быть освобождена

```
int MPI_Type_free(MPI_Datatype *type);
```

Общий способ конструирования производных типов данных

Пример:

```
double a; /* адрес 24 */
double b; /* адрес 40 */
int    n; /* адрес 48 */
int BlockLens[3];
MPI_Aint Offsets[3];
MPI_Datatype Types[3];
MPI_Datatype NewType;
MPI_Aint addr, start_addr;
/* кол-во значений */
BlockLens[0]=BlockLens[1]=BlockLens[2]=1;
/* типы */
Types[0]=MPI_DOUBLE; Types[1]=MPI_DOUBLE;
Types[2]=MPI_INT;
/* смещения */
MPI_Adress(&a,&start_addr); Offsets[0] = 0;
MPI_Adress(&b,&addr); Offsets[1] = addr-start_addr;
MPI_Adress(&n,&addr); Offsets[2] = addr-start_addr;
MPI_Type_struct(3, BlockLens, Offsets, Types, &NewType);
MPI_Type_commit(&NewType);
```

Характеристики производного типа данных…

- Размер типа
  - Число байт, которые занимают данные

```
int MPI_Type_size(MPI_Datatype datatype,
  MPI_Aint *size);
```

- Нижняя граница
  - lb(Typemap) = min ($disp_j$)

```
int MPI_Type_lb(MPI_Datatype datatype,
  MPI_Aint *displacement);
```

- Верхняя граница
  - ub(Typemap) = max ($disp_j$ + sizeof ($type_j$))

```
int MPI_Type_ub(MPI_Datatype datatype,
  MPI_Aint *displacement);
```

- Протяженность типа
  - extent(Typemap) = ub – lb + pad

```
int MPI_Type_extent(MPI_Datatype datatype,
  MPI_Aint *extent);
```

Характеристики производного типа данных

Протяженность типа приводится в размеру, кратному длине первого значения типа (с тем, чтобы для производного типа можно было бы образовывать вектора)

Пример

Карта типа

```
{(MPI_DOUBLE,0),(MPI_DOUBLE,16),(MPI_INT,24)}
```

- размер20 (int 4)
- нижняя граница 0
- верхняя граница 28
- протяженность 32

Дополнительные способы конструирования производных типов данных

- Непрерывный
- Векторный
- Н-Векторный
- Индексный
- Н-Индексный
- Упакованный

Непрерывный способ конструирования производных типов данных

Это один из самых простых способов конструирования производных типов данных из count значений существующего типа со смещениями, увеличивающимися на протяженность oldtype (исходного типа).

int MPI_Type_contiguous (count, oldtype, newtype)

Векторный способ конструирования производных типов данных

Это более общий способ формирования непрерывного типа, который допускает регулярные промежутки между значениями (длина промежутка должна быть кратной протяженности исходного типа данных).

```
int MPI_Type_vector(int count, int blocklen,
  int stride, MPI_Datatype old_type,
  MPI_Datatype *newtype);
```

Пример

Вектор $x = (x_1,\ldots,x_{20})$ с элементами типа double.

Необходимо использовать только четные элементы.

```
MPI_Type_vector(10, 1, 2, MPI_DOUBLE, &newtype);
```

H-Векторный способ конструирования производных типов данных

Подобен векторному способу конструирования, но расстояние между элементами задается в байтах.

```
int MPI_Type_hvector(int count, int blocklen,
  MPI_Aint stride, MPI_Datatype old_type,
  MPI_Datatype *newtype);
```

Индексный способ конструирования производных типов данных

При создании этого типа используется массив смещений исходного типа данных; единица смещения соответствует протяженности исходного типа данных.

```
int MPI_Type_indexed(int count, int blocklens[],
    int indices[], MPI_Datatype old_type,
    MPI_Datatype *newtype);
```

H-Индексный способ конструирования производных типов данных

Подобен индексному, но смещения задаются в байтах.

```
int MPI_Type_hindexed(int count, int blocklens[],
   int indices[], MPI_Datatype old_type,
   MPI_Datatype *newtype);
```

Упакованный способ конструирования производных типов данных…

При данном способе пересылаемые данные предварительно собираются (пакуются) в дополнительном буфере; при получении сообщения выполняется обратная операция – распаковка данных.

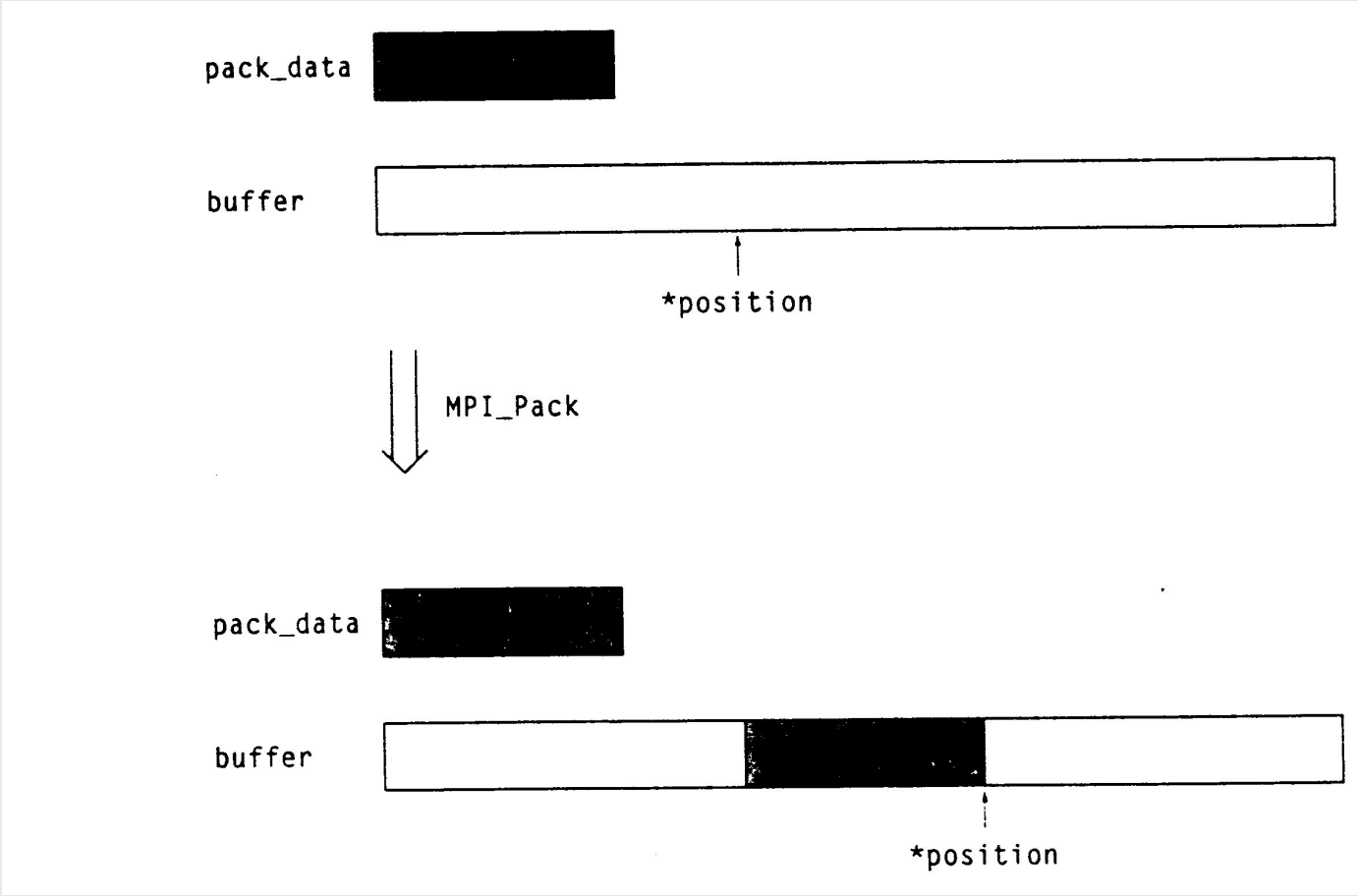
int **MPI_Pack**(void* sdatbuf, int sdatcount, MPI_Datatype sdattype, void *packbuf, int packsize, int *packpos, MPI_Comm comm)

int **MPI_Unpack**(void* packbuf, int packsize, int *packpos, void *rdatbuf, int rdatcount, MPI_Datatype rdattype, MPI_Comm comm)

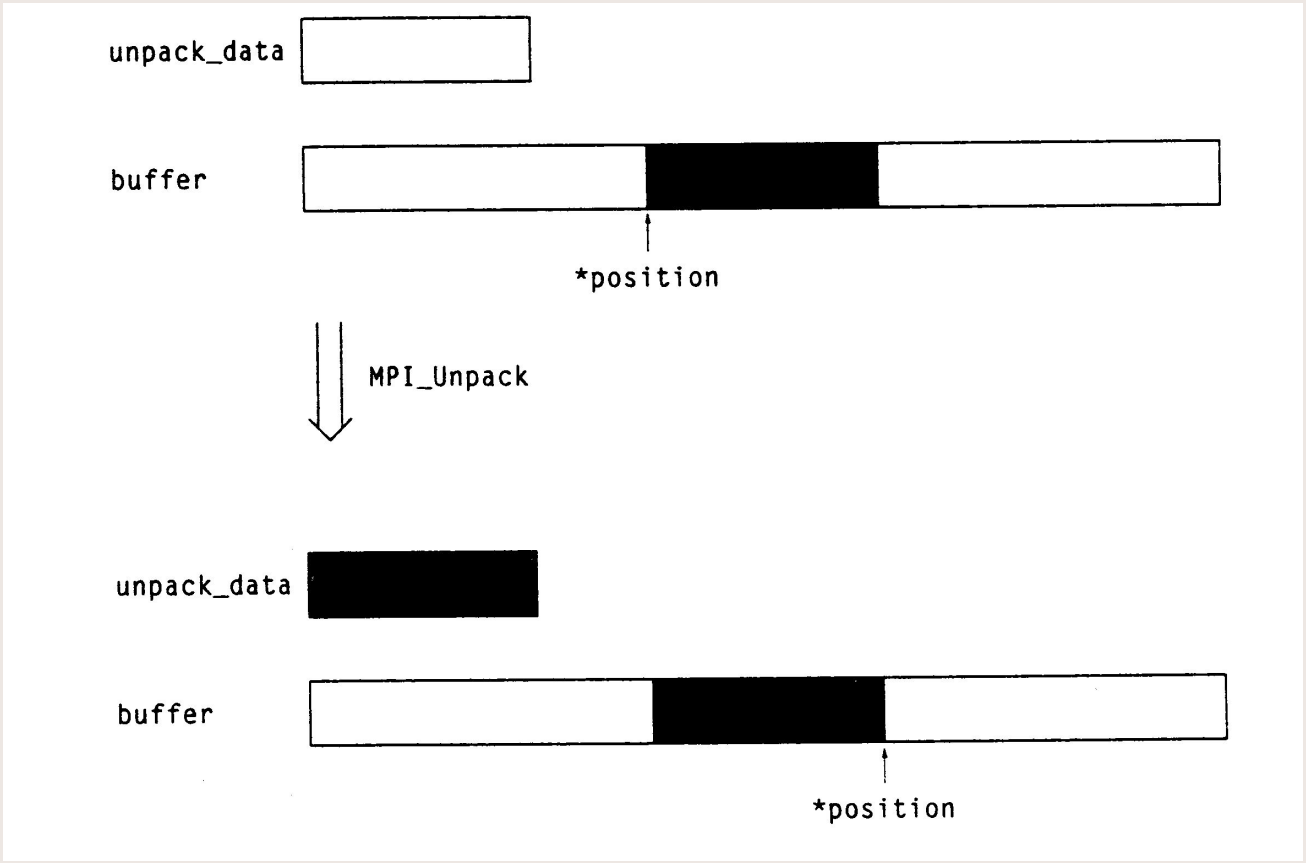
где

- (sdatbuf, sdatcount, sdattype) – данные, упаковываемые для передачи,
- (packbuf, packsize, packpos) – буфер для упаковки или распаковки,
- (rdatbuf, rdatcount, rdattype) – данные, распаковываемые после приема сообщения

Упакованный способ конструирования производных типов данных…



Упакованный способ конструирования производных типов данных…



Упакованный способ конструирования производных типов данных

- Значение packpos указывает позицию буфера для упаковки или распаковки; начальное значение – 0, дальнейшее формирование выполняется функциями MPI_Pack и MPI_Unpack,
- Для завершения упаковки или распаковки функция MPI_Commit не используется,
- В функциях передачи данных для типа используется идентификатор MPI_PACKED

Правила соответствия типов

Сигнатура типа отправляемого сообщения должна являться начальным отрезком (или совпадать) сигнатуры типа принимаемого сообщения, т.е.

Сигнатура отправляемого сообщения

$$\{(stype_0,\ldots,stype_{n-1}\}$$

Сигнатура принимаемого сообщения

$$\{(rtype_0,\ldots,rtype_{m-1}\}$$

$\Rightarrow$ $n \le m$

$stype_i == rtype_i$, $0 \le i \le n-1$

! Для коллективных операций сигнатуры типов во всех процессах должны совпадать

Рекомендации по выбору способа конструирования производных типов данных

- Если пересылаемые данные образуют непрерывный блок значений одного и того же базового типа – создание производного типа не является необходимым
- Если данные занимают несмежные области памяти разного размера – целесообразно создание нового типа (в зависимости от количества передач возможно использование упаковки/распаковки данных)
- Если разрывы (промежутки) между областями значений имеют постоянный размер, наиболее подходящий способ конструирования – векторный метод; при промежутках различного размера выгодным является использование индексного способа конструирования

Систематика процессов (коммуникаторы и и группы)

- *Группа* – набор процессов; каждый процесс в пределах группы идентифицируется уникальным номером (*рангом*)
- *Коммуникатор* – группа процессов с определенным *контекстом* для применения функций MPI (значениями параметров, топологией и др.)
- Для организации структурности (модульности) процесса вычислений и снижения сложности программирования программистом могут формироваться из существующих процессов новые группы и коммуникаторы

Методы работы с группами

- Формирование группы из процессов существующего коммуникатора

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group);
```

- Формирование группы из конкретного набора процессов существующей группы

```
int MPI_Group_incl(MPI_Group oldgroup, int n,
   int *ranks, MPI_Group *newgroup);
```

- Формирование группы путем исключения конкретного набора процессов существующей группы

```
int MPI_Group_excl(MPI_Group oldgroup, int n,
   int *ranks, MPI_Group *newgroup);
```

- Освобождение дескриптора группы (группа не разрушается, если на группу есть ссылка в коммуникаторе)

```
int MPI_Group_free(MPI_Group group);
```

после выполнения функции `group==MPI_GROUP_NULL`

(всего 12 функций)

Методы работы с коммуникаторами…

- Создание коммуникатора по группе

```
int MPI_Comm_create(MPI_Comm comm, MPI_Group
  group, MPI_Comm *newcomm);
```

- Создание копии коммуникатора

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm
  *newcomm);
```

- Удаление коммуникатора

```
int MPI_Comm_free(MPI_Comm comm);
```

после выполнения функции comm==**MPI_COMM_NULL**

(всего 11 функций)

Методы работы с коммуникаторами…

Пример:

```
MPI_Group world_group, worker_group;
MPI_Comm workers;
int ranks[1];
ranks[0] = 0;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);
MPI_Group_excl(world_group, 1, ranks, &worker_group);
MPI_Comm_create(MPI_COMM_WORLD,worker_group,&workers);
...
MPI_Group_free(worker_group);
MPI_Comm_free(workers);
```

Методы работы с коммуникаторами…

- Одновременное создание нескольких коммуникаторов

```
int MPI_Comm_split( MPI_Comm oldcomm, int
  commnum, int newrank, MPI_Comm *newcomm );
```

где

- `commnum` – номер создаваемого коммуникатора,
- `newrank` – ранг процесса в новом коммуникаторе

! Операция является коллективной для процессов используемого коммуникатора и должна выполниться в каждом процессе

Методы работы с коммуникаторами

- Пример для использования **MPI_Comm_split**

```
Формирование логической структуры процессов в виде
  двумерной решетки (в примере будет показано
  создание коммуникаторов для каждой строки
  создаваемой топологии). Пусть p=q*q есть общее
  количество процессов
MPI_Comm comm;
int rank, row;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
row = rank/q;
MPI_Comm_split(MPI_COMM_WORLD,row,rank,&comm);

 Например, если p=9, то процессы (0,1,2) образуют
  первый коммуникатор, процессы (3,4,5) - второй и т.
  д.
```

Литература

1. “MPI для начинающих” Автор: Илья Евсеев. Учебное пособие плюс примеры. *On-line: http://www2.sscc.ru/Litera/il/*
2. “Parallel Programming With MPI” Автор: Peter Pacheco
3. “MPI: The Complete Reference” Авторы: Marc Snir, Steve Otto, Steve Huss-Lederman, David Walker, Jack Dongarra
4. “Parallel Processing Online Help With MPI” *On-line: http://www.coe.uncc.edu/~abw/parallel/mpi/*
5. “Using MPI. (Portable Parallel Programming with the Message-Passing Interface)” Авторы: W.Group, E.Lusk, A.Skjellum *On-line:* http://www.mcs.anl.gov/mpi/index.html

Вопросы для обсуждения

- Рекомендации по использованию разных способов конструирования типов данных
- Целесообразность использования дополнительных коммуникаторов

Задания для самостоятельной работы

- Создание производных типов данных для задач линейной алгебры (типы для строк, столбцов, диагоналей, горизонтальных и вертикальных полос, блоков матриц)
- Создание коммуникаторов для процессов, располагающихся в столбцах прямоугольной решетки

Заключение

- Методы конструирования производных типов данных
- Методы создания новых коммуникаторов

Следующая тема

- Методы разработки параллельных программ при использования интерфейса передачи сообщений MPI-3