Санкт-Петербургский государственный университет
кафедра вычислительной физики

С.А.Немнюгин

Лабораторная работа 0.3
Знакомство с процедурами коллективного обмена MPI

Методические материалы к курсу «Средства программирования для многопроцессорных вычислительных систем»

Intel Multicore Curriculum Initiative

Санкт-Петербург
2007

Коллективные обмены

При выполнении коллективного обмена сообщение пересылается от одного процесса нескольким или наоборот, один процесс собирает данные от нескольких процессов. MPI поддерживает такие виды коллективного обмена, как *широковещательная передача*, *операции приведения* (*редукции*), *распределение* и *сбор* данных и т. д.

Коллективные обмены характеризуются следующим:

- коллективные обмены не могут взаимодействовать с двухточечными. Коллективная передача, например, не может быть перехвачена двухточечной подпрограммой приема;
- коллективные обмены могут выполняться как с синхронизацией, так и без нее;
- все коллективные обмены являются блокирующими для инициировавшего их обмена;
- теги сообщений назначаются системой.

В коллективном обмене участвует каждый процесс из некоторой области взаимодействия. Можно организовать обмен и в подмножестве процессов, для этого имеются средства создания новых областей взаимодействия и соответствующих им коммуникаторов.

Широковещательная рассылка

Широковещательная рассылка выполняется выделенным процессом, который называется *главным* (root). Все остальные процессы, принимающие участие в обмене, получают по одной копии сообщения от главного процесса (рис. 1).

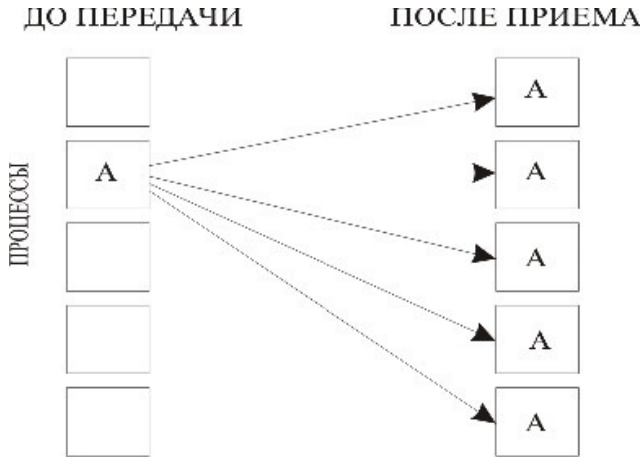

Рис. 1. Широковещательная рассылка

Выполняется широковещательная рассылка с помощью подпрограммы `MPI_Bcast` (см. Методическое пособие).

Операции редукции

Операции редукции относятся к категории глобальных вычислений. В глобальной операции приведения к данным от всех процессов из заданного коммуникатора применяется операция `MPI_Reduce` (рис. 2).

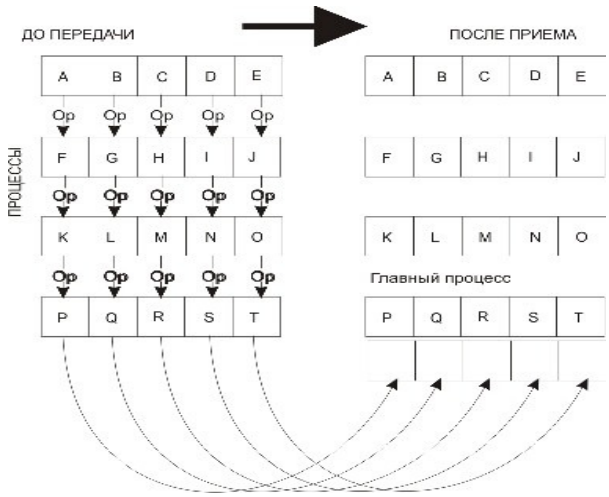

Рис. 2. Глобальная операция приведения

Аргументом операции приведения является массив данных — по одному элементу от каждого процесса. Результат такой операции — единственное значение.

Создание группы процессов

Для организации коллективных обменов на подмножестве процессов создают группу и соответствующий ей коммуникатор. *Группой* называют упорядоченное множество процессов. Каждому процессу в группе сопоставлен свой ранг. Операции с группами могут выполняться отдельно от операций с коммуникаторами, но в операциях обмена используются только коммуникаторы. В MPI имеется специальная предопределенная пустая группа `MPI_GROUP_EMPTY`.

Коммуникаторы бывают двух типов: *интракоммуникаторы* — для операций внутри одной группы процессов и *интеркоммуникаторы* — для двухточечного обмена между двумя группами процессов.

В MPI-программах чаще используются интракоммуникаторы. Интракоммуникатор включает экземпляр группы, контекст обмена для всех его видов, а также, возможно, виртуальную топологию и другие атрибуты.

Созданию нового коммуникатора предшествует создание соответствующей группы процессов. Операции создания групп аналогичны математическим операциям над множествами:

- *объединение* — к процессам первой группы добавляются процессы второй группы, не принадлежащие первой;
- *пересечение* — в новую группу включаются все процессы, принадлежащие двум группам одновременно. Ранги им назначаются как в первой группе;
- *разность* — в новую группу включаются все процессы первой группы, не входящие во вторую группу. Ранги назначаются как в первой группе.

Новую группу можно создать только из уже существующих групп. Базовая группа, из которой формируются все другие группы, связана с коммуникатором `MPI_COMM_WORLD`.

Доступ к группе `group`, связанной с коммуникатором `comm` можно получить, обратившись к подпрограмме `MPI_Comm_group`.

В MPI имеются подпрограммы-конструкторы новых групп (см. Методическое пособие). Есть и деструктор - `MPI_Group_free`.

Создание коммуникатора — коллективная операция и соответствующая подпрограмма должна вызываться всеми процессами коммуникатора. Подпрограмма `MPI_Comm_dup` дублирует уже существующий коммуникатор.

Подпрограмма `MPI_Comm_create` создает новый коммуникатор из подмножества процессов другого коммуникатора.

Вызов этой подпрограммы должны выполнить все процессы из старого коммуникатора, даже если они не входят в группу `group`, с одинаковыми аргументами. Данная операция применяется только к интракоммуникаторам. Она позволяет выделять подмножества процессов со своими областями взаимодействия, если, например, требуется уменьшить "зернистость" параллельной программы.

Лабораторная работа

В заданиях лабораторной работы 0.3 предлагается дописать предлагаемые фрагменты программ на языке С, написанные с использованием процедур MPICH 1.2.7. Пропущенные фрагменты обозначены многоточием.
Необходимый для выполнения данной лабораторной работы справочный материал можно найти на стр. 37 – 49 *методического пособия* «Средства программирования для многопроцессорных вычислительных систем».

Задание 1

В исходном тексте программы на языке С пропущены вызовы процедур широковещательной рассылки. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 char data[24];
 int myrank, count = 25;
 MPI_Status status;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
{
 strcpy(data, "Hi, Parallel Programmer!");
 …
 printf("send: %s\n", data);
}
else
{
 …
 printf("received: %s\n", data);
}
 MPI_Finalize();
 return 0;
}
```

Задание 2

В программе на языке С предполагается, что три численных значения, введенных с клавиатуры, пересылаются широковещательной рассылкой всем прочим процессам. Вызовы подпрограмм широковещательной рассылки пропущены. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
 int myrank;
 int root = 0;
 int count = 1;
 float a, b;
 int n;

 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
{
printf("Enter a, b, n\n");
scanf("%f %f %i", &a, &b, &n);
…
}
else
{
…
printf("%i Process got %f %f %i\n", myrank, a, b, n);
}
 MPI_Finalize();
 return 0;
}
```

Задание 3

В программе на языке С создается новый коммуникатор, а затем сообщения между процессами, входящими в него, пересылаются широковещательной рассылкой. Вызовы подпрограмм создания новой группы процессов (на 1 меньше, чем полное количество запущенных на выполнение процессов) и нового коммуникатора пропущены. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 char message[24];
 MPI_Group MPI_GROUP_WORLD;
 MPI_Group group;
 MPI_Comm fcomm;
 int size, q, proc;
 int* process_ranks;
 int rank, rank_in_group;

 MPI_Status status;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 printf("New group contains processes:");
 q = size - 1;
 process_ranks = (int*) malloc(q*sizeof(int));
 for (proc = 0; proc < q; proc++)
{
 process_ranks[proc] = proc;
 printf("%i ", process_ranks[proc]);
 }
 printf("\n");
 …
 if (fcomm != MPI_COMM_NULL) {
 MPI_Comm_group(group, &fcomm);
 MPI_Comm_rank(fcomm, &rank_in_group);
 if (rank_in_group == 0) {
 strcpy(message, "Hi, Parallel Programmer!");
 MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
 printf("0 send: %s\n", message);
 }
else
{
 MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
 printf("%i received: %s\n", rank_in_group, message);
}
 MPI_Comm_free(&fcomm);
 MPI_Group_free(&group);
}
 MPI_Finalize();
 return 0;}
```

Задание 4

В программе на языке С сначала создается подгруппа, состоящая из процессов с рангами 1, 3, 5 и 7, и соответствующий ей коммуникатор. Затем выполняется редукция (суммирование) по процессам, входящим в новую группу. Вызов подпрограммы редукции и некоторые другие важные фрагменты пропущены. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int myrank, i;
 int count = 5, root = 1;
 MPI_Group MPI_GROUP_WORLD, subgroup;
 int ranks[4] = {1, 3, 5, 7};
 MPI_Comm subcomm;
 int sendbuf[5] = {1, 2, 3, 4, 5};
 int recvbuf[5];

 MPI_Init(&argc, &argv);
 MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
 MPI_Group_incl(MPI_GROUP_WORLD, 4, ranks, &subgroup);
 MPI_Group_rank(subgroup, &myrank);
 …

if(myrank != MPI_UNDEFINED)
{
MPI_Reduce(&sendbuf, &recvbuf, count, MPI_INT, MPI_SUM, root,
subcomm);

if(myrank == root) {
printf("Reduced values");
for(i = 0; i < count; i++){
printf(" %i ", recvbuf[i]);}
}
printf("\n");

MPI_Comm_free(&subcomm);
MPI_Group_free(&MPI_GROUP_WORLD);
 …
}
 MPI_Finalize();
 return 0;
}
```