Лабораторная работа 4

Знакомство с процедурами коллективного обмена MPI

Цель работы:

Приобрести навыки составления MPI-программ. Изучить функции коллективного обмена MPI, такие как широковещательная передача, операции приведения (редукции), распределение и сбор данных.

Задачи работы:

1) Изучить теоретические основы работы функций коллективного обмена MPI.
2) Выполнить общие задания 1-4 лабораторной работы и дополнительное задание в соответствии со своим вариантом.
3) Ответить на вопросы.

Теория

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*, *сбор данных* и т.д.

Коллективные обмены характеризуются следующим:

- коллективные обмены не могут взаимодействовать с двухточечными обменами (коллективная передача, например, не может быть перехвачена двухточечной подпрограммой приема);
- коллективные обмены могут выполняться как с синхронизацией, так и без неё;
- 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 (рисунок 1).

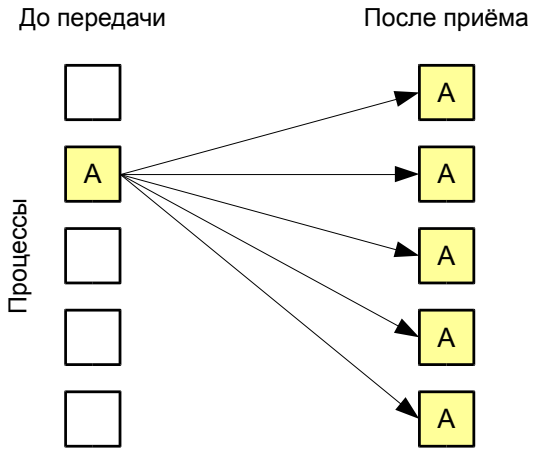

Рисунок 1 – Широковещательная рассылка

Выполняется широковещательная рассылка с помощью функции `MPI_Bcast(…)`. Прототип функции `MPI_Bcast(…)` следующий:

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
              int root, MPI_Comm comm)
```

Параметры функции одновременно являются входными и выходными:

- **`buffer`** – адрес буфера;
- **`count`** – количество элементов данных в сообщении;
- **`datatype`** – тип данных MPI;
- **`root`** – ранг главного процесса, выполняющего широковещательную рассылку;
- **`comm`** – коммуникатор.

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(…)` (рисунок 2).

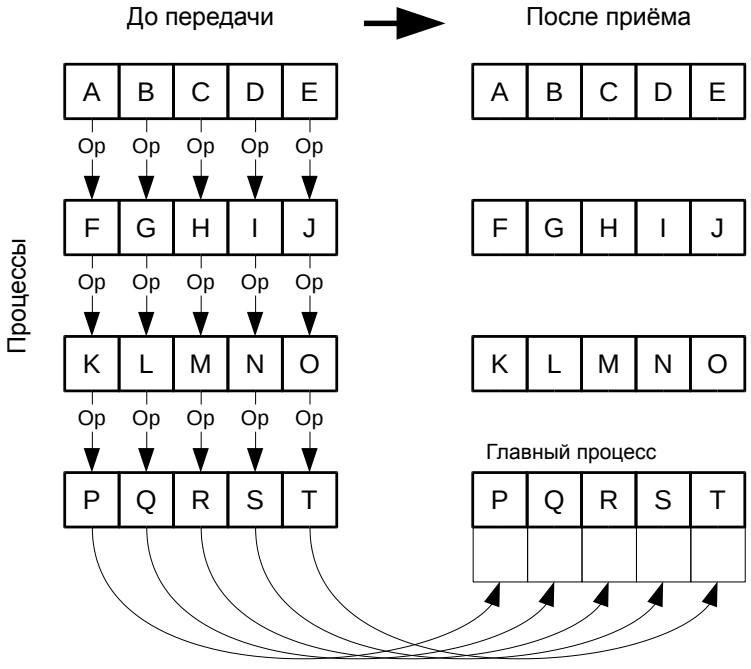

Рисунок 2 – Глобальная операция приведения

Аргументом операции приведения является массив данных – по одному элементу от каждого процесса. Результат такой операции – единственное значение. Прототип функции `MPI_Reduce(…)` следующий:

```
int MPI_Reduce(void *buf, void *result, int count,
               MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm)
```

Входные параметры:

- **`buf`** – адрес буфера передачи;
- **`count`** – количество элементов в буфере передачи;
- **`datatype`** – тип данных в буфере передачи;
- **`op`** – операция приведения;
- **`root`** – ранг главного процесса;
- **`comm`** – коммуникатор.

Выходной параметр:

- **`result`** – адрес начала буфера результатов (используется только в процессе-получателе `root`);

`MPI_Reduce(…)` применяет операцию приведения к операндам из `buf`, а результат каждой операции помещается в буфер результата `result`. `MPI_Reduce(…)` должна вызываться всеми процессами в коммуникаторе `comm`, а аргументы `count`, `datatype` и `op` в этих вызовах должны совпадать.

Предопределенные операции приведения представлены в таблице 1.

Таблица 1 – Предопределенные операции приведения MPI

Операция	Описание
`MPI_MAX`	Определение максимальных значений элементов одномерных массивов целого или вещественного типа
`MPI_MIN`	Определение минимальных значений элементов одномерных массивов целого или вещественного типа
`MPI_SUM`	Вычисление суммы элементов одномерных массивов целого, вещественного или комплексного типа
`MPI_PROD`	Вычисление поэлементного произведения одномерных массивов целого, вещественного или комплексного типа
`MPI_LAND`	Логическое "И"
`MPI_BAND`	Битовое "И"
`MPI_LOR`	Логическое "ИЛИ"
`MPI_BOR`	Битовое "ИЛИ"
`MPI_LXOR`	Логическое исключающее "ИЛИ"
`MPI_BXOR`	Битовое исключающее "ИЛИ"
`MPI_MAXLOC`	Максимальные значения элементов одномерных массивов и их индексы
`MPI_MINLOC`	Минимальные значения элементов одномерных массивов и их индексы
`MPI_LAND`	Логическое "И"

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`**.

Имеются также стандартные коммуникаторы:

- **`MPI_COMM_SELF`** – коммуникатор, содержащий только вызывающий процесс;
- **`MPI_COMM_NULL`** – пустой коммуникатор.

Доступ к группе `group`, связанной с коммуникатором `comm` можно получить, обратившись к функции `MPI_Comm_group(…)`, прототип которой выглядит так:

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

Выходной параметр – группа. Для выполнения операций с группой к ней сначала необходимо получить доступ.

Пометить коммуникатор `comm` для удаления можно с помощью следующей функции, прототип которой выглядит следующим образом:

```
int MPI_Comm_free(MPI_Comm *comm)
```

Обмены, связанные с этим коммуникатором, завершаются обычным образом, а сам коммуникатор удаляется только после того, как на него не будет активных ссылок. Данная операция может применяться к коммуникаторам интра- и интеркоммуникаторам.

В MPI имеются подпрограммы-конструкторы новых групп:

- Создание новой группы `newgroup` из `n` процессов, входящих в группу `oldgroup` осуществляется с помощью функции `MPI_Group_incl(…)`, прототип которой выглядит так:

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

  Ранги процессов содержатся в массиве `ranks`. В новую группу войдут процессы с рангами `ranks[0], …, ranks[n-1]`, причем рангу `i` в новой группе соответствует ранг `ranks[i]` в старой группе. При `n = 0` создается пустая группа `MPI_GROUP_EMPTY`. С помощью данной подпрограммы можно не только создать новую группу, но и изменить порядок процессов в старой группе.

- Создание группы `newgroup` исключением из исходной группы (`group`) процессы с рангами `ranks[0], …, ranks[n-1]` осуществляется с помощью функции `MPI_Group_excl(…)`, прототип которой выглядит так:

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

  При `n = 0` новая группа тождественна старой.

- Создание группы `newgroup` из группы `group` добавлением в нее `n` процессов, ранг которых указан в массиве `ranks` осуществляется с помощью функции `MPI_Group_range_incl(…)`, прототип которой выглядит так:

```
int MPI_Group_range_incl(MPI_Group oldgroup, int n,
            int ranks[][3], MPI_Group *newgroup)
```

  Массив `ranks` состоит из целочисленных триплетов вида «`(первый_1, последний_1, шаг_1), …, (первый_n, последний_n, шаг_n)`». В новую группу войдут процессы с рангами (по первой группе) «`первый_1, первый_1 + шаг_1, …`».

- Создание группы `newgroup` из группы `group` исключением из нее `n` процессов, ранг которых указан в массиве `ranks` осуществляется с помощью функции `MPI_Group_range_excl(…)`, прототип которой выглядит так:

```
int MPI_Group_range_excl(MPI_Group group, int n,
            int ranks[][3], MPI_Group *newgroup)
```

  Массив `ranks` устроен так же, как аналогичный массив в подпрограмме `MPI_Group_range_incl`.

- Создание новой группы `newgroup` из разности двух групп `group1` и `group2` осуществляется с помощью функции `MPI_Group_difference(…)`, прототип которой выглядит так:

```
int MPI_Group_difference(MPI_Group group1, MPI_Group group2,
            MPI_Group *newgroup)
```

- Создание новой группы `newgroup` из пересечения групп `group1` и `group2` осуществляется с помощью функции `MPI_Group_intersection(…)`, прототип которой выглядит так:

```
int MPI_Group_intersection(MPI_Group group1, MPI_Group group2,
            MPI_Group *newgroup)
```

- Создание группы `newgroup` объединением групп `group1` и `group2` осуществляется с помощью функции `MPI_Group_union(…)`, прототип которой выглядит так:

```
int MPI_Group_union(MPI_Group group1, MPI_Group group2,
            MPI_Group *newgroup)
```

Есть и деструктор – функция `MPI_Group_free(…)`, прототип которой выглядит так:

```
int MPI_Group_free(MPI_Group *group)
```

Определить количество процессов `size` в группе `group` можно с помощью функции `MPI_Group_size(…)`, прототип которой выглядит следующим образом:

```
int MPI_Group_size(MPI_Group group, int *size)
```

Определить ранг `rank` процесса в группе `group` можно с помощью функции `MPI_Group_rank(…)`, прототип которой выглядит следующим образом:

```
int MPI_Group_rank(MPI_Group group, int *rank)
```

Если процесс не входит в указанную группу, возвращается значение **MPI_UNDEFINED**.

Создание коммуникатора – коллективная операция и соответствующая функция должна вызываться всеми процессами коммуникатора. Функция `MPI_Comm_dup(…)` дублирует уже существующий коммуникатор `oldcomm`. Прототип функции `MPI_Comm_dup(…)` выглядит так:

```
int MPI_Comm_dup(MPI_Comm oldcomm, MPI_Comm *newcomm)
```

В результате вызова данной подпрограммы создается новый коммуникатор `newcomm` с той же группой процессов, с теми же атрибутами, но с другим контекстом. Подпрограмма может применяться как к интра-, так и к интеркоммуникаторам.

Подпрограмма `MPI_Comm_create(…)` создает новый коммуникатор `newcomm` из подмножества процессов `group` другого коммуникатора `oldcomm`. Прототип функции `MPI_Comm_create(…)` выглядит так:

```
int MPI_Comm_create(MPI_Comm oldcomm, MPI_Group group,
            MPI_Comm *newcomm)
```

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. Если одновременно создаются

несколько коммуникаторов, они должны создаваться в одной последовательности всеми процессами.

Трансляция и выполнение MPI-программ

Для трансляции и компоновки программ на языке *C++* используется команда **mpiCC**, для трансляции программ на языке *C* – команда **mpicc**. Пример применения команды:

```
mpicc –o program source.c
```

где **program** – желаемое имя исполняемого файла программы, **source.c** – имя файла с исходным кодом программы.

Для выполнения MPI-программ используется загрузчик приложений **mpirun**. Он запускает указанное количество копий программы. Команда запуска:

```
mpirun –np X [ключи MPI] program [ключи и аргументы программы]
```

где **X** – число запускаемых процессов, **program** – имя исполняемого файла программы.

Например, после развёртывания кластера PelicanHPC, строка запуска программы может выглядеть так:

```
mpirun –np 10 --hostfile /home/user/tmp/bhosts program
```

т.е. будет запущено **10** процессов программы **program**, IP-адреса вычислительных узлов кластера будут получены из файла **bhosts**, до которого указан полный путь.

Лабораторная работа

В заданиях лабораторной работы предлагается дописать пропущенные фрагменты программ на языке *C* (программы написаны с использованием процедур MPICH 1.2.7). Пропущенные фрагменты обозначены многоточием.

Задание 1

В исходном тексте программы на языке *C* 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 %i: %s\n", myrank, data);
  }
  else
  {
    ....
    printf("Received %i: %s\n", myrank, data);
  }
  MPI_Finalize();
  return 0;
}
```

Задание 2

В программе на языке *C* 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 *C* 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 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");

  MPI_Comm_group(....);
  MPI_Comm_create(....);

  if (fcomm != MPI_COMM_NULL)
  {
    MPI_Comm_group(fcomm, &group);
    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 *C* 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);

  MPI_Comm_create(....);

  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_Group_free(....);
  }
  MPI_Finalize();
  return 0;
}
```

Задания по вариантам

1) Создайте программу, используя функцию широковещательной рассылки `MPI_Bcast(…)`, реализующую алгоритм передачи данных от 0 процесса всем остальным.
2) Создайте программу, используя функцию широковещательной рассылки `MPI_Bcast(…)`, реализующую алгоритм передачи данных от каждого процесса всем остальным.
3) Нулевой процесс «загадывает» случайное число в диапазоне от 1 до 9 включительно и отправляет его всем остальным процессам. Если принятое число совпадает с номером процесса, то он отвечает сообщением нулевому процессу.
4) Каждый процесс, включая нулевой, «загадывает» случайное число в диапазоне от 1 до 9 включительно. Нулевой процесс посылает своё число всем остальным, и если оно совпадает с числом, загаданным данным процессом, то он отправляет это число и свой номер, иначе отправляет 0.
5) Используя функции для создания коммуникаторов и коммуникационную функцию `MPI_Bcast(…)` реализуйте алгоритм передачи от всех чётных процессов своего номера всем нечётным процессам.
6) Используя функции для создания коммуникаторов и коммуникационную функцию `MPI_Bcast(…)` реализуйте алгоритм передачи от всех нечётных процессов своего номера всем чётным процессам.
7) Используя функции для создания коммуникаторов, реализуйте следующий алгоритм: на нулевом процессоре задана переменная `A`, на первом процессоре – `B`; с помощью коммуникационной функции `MPI_Bcast(…)` передайте переменную `A` – четным процессорам, переменную `B` – нечетным.
8) Напишите программу, используя коммуникационную функцию `MPI_Bcast(…)` и глобальную вычислительную функцию `MPI_Reduce(…)`, реализующую алгоритм суммирования ряда чисел.
9) Напишите программу, используя коммуникационную функцию `MPI_Bcast(…)` и глобальную вычислительную функцию `MPI_Reduce(…)`, реализующую алгоритм произведение ряда чисел.
10) Напишите программу, используя коммуникационную функцию `MPI_Bcast(…)` и глобальную вычислительную функцию `MPI_Reduce(…)`, реализующую алгоритм суммирования ряда чисел.
11) Напишите программу, используя коммуникационную функцию `MPI_Bcast(…)` и глобальную вычислительную функцию `MPI_Reduce(…)`, реализующую алгоритм поиска максимального элемента в одномерном массиве целых чисел. Первоначальный массив формируется нулевым процессом из псевдослучайных чисел.
12) Напишите программу, используя коммуникационную функцию `MPI_Bcast(…)` и глобальную вычислительную функцию `MPI_Reduce(…)`, реализующую алгоритм поиска минимального элемента в одномерном массиве целых чисел. Первоначальный массив формируется нулевым процессом из псевдослучайных чисел.

13) (*) Создайте программу, используя коммуникационную функцию `MPI_Gather(…)`, реализующую алгоритм передачи частей массива от всех процессоров на нулевой процессор.
14) (*) Создайте программу, используя коммуникационную функцию `MPI_Allgather(…)`, реализующую алгоритм передачи частей массива от всех процессоров на все процессора.
15) (*) Создайте программу, используя коммуникационную функцию `MPI_Scatter(…)`, реализующую алгоритм передачи частей массива от нулевого процессора на все процессора.

Вопросы

1) Какие виды коллективного обмена поддерживает MPI?
2) Каким образом можно организовать обмен в подмножестве процессов?
3) Какой функцией выполняется широковещательная рассылка?
4) Для чего нужны операции редукции?
5) Что такое группа процессов и зачем она создаётся?
6) Для чего может потребоваться создание нового коммуникатора?