Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

Параллельное программирование для многопроцессорных систем с разделяемой памятью

04 Лекция
Расширенные возможности MPI

Сысоев А.В.

Содержание

- Неблокирующие коллективные операции
  - Общее описание
  - Передача данных
  - Операции сокращения данных
  - Разделение и сбор данных
  - Всеобщие обмены
  - Синхронизация вычислений
- Создание и управление процессами
  - Общее описание
  - Динамическая модель процессов
  - Управление процессами
  - Установка соединений

Неблокирующие коллективные операции

- Передача данных
- Операции сокращения данных
- Разделение и сбор данных
- Всеобщие обмены
- Синхронизация вычислений

Неблокирующие коллективные обмены… Общее описание

- Неблокирующие коллективные операции обладают потенциальными преимуществами неблокирующих двусторонних операций, а также оптимизированным выполнением и планированием сообщений
- Одним из способов реализации является выполнение блокирующей коллективной операции на отдельном потоке
- Неблокирующий коллективный обмен часто приводит к лучшей производительности (избегает переключений контекста, накладных расходов планировщика и управления потоками)
- Как и блокирующие, неблокирующие коллективные операции считаются завершенными, когда локальная часть операции завершена

Неблокирующие коллективные обмены… Общее описание

- Окончание выполнения операции не означает, что другие процессы завершили или даже начали операцию (если описание операции не говорит об обратном)
- Окончание выполнения определенной неблокирующей коллективной операции также не означает завершение любой другой начатой неблокирующей коллективной или парной операции независимо от порядка их вызова
- Пользователи должны иметь в виду, что реализациям стандарта MPI разрешается, но не требуется (за исключением MPI_IBARRIER) синхронизировать процессы при выполнении неблокирующей коллективной операции

Неблокирующие коллективные обмены… Общее описание

- В отличие от двусторонних операций, неблокирующие коллективные операции не являются точными неблокирующими аналогами блокирующих коллективных операций
- Все процессы должны вызывать коллективные операции (блокирующие и неблокирующие) в одном и том же порядке (для указанного в операциях коммуникатора)
- Когда процесс вызывает коллективную операцию, все другие процессы в коммуникаторе должны в конечном счете вызвать ту же коллективную операцию, не вызывая до этого других коллективных операций с тем же коммуникатором

Неблокирующие коллективные обмены… Передача данных

```
int MPI_Ibcast(void *buf, int count, MPI_Datatype type,
  int root, MPI_Comm comm, MPI_Request *request);
```

- **buf** – адрес буфера памяти, содержащего данные передаваемого сообщения
- **count** – число элементов данных в сообщении
- **type** – тип элементов данных в сообщении
- **root** – номер процесса, выполняющего передачу данных
- **comm** – коммуникатор, по которому передаются данные
- **request** – дескриптор операции

- Функция **MPI_Ibcast()** выполняет передачу данных из буфера **buf**, содержащего **count** элементов типа **type**, от процесса с номером **root** процессам, входящим в коммуникатор **comm**

Неблокирующие коллективные обмены… Редукция данных

- Чтобы “редуцировать” данные со всех процессов на выбранном

```
int MPI_Ireduce(void *sendbuf, void *recvbuf, int count,
  MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm,
  MPI_Request *request);
```

- **sendbuf** – буфер памяти с передаваемым сообщением
- **recvbuf** – буфер памяти с результирующим сообщением (только для корневого процесса)
- **count** – число элементов данных в сообщении
- **type** – тип элементов данных в сообщении
- **op** – операция, которая должна быть выполнена
- **root** – номер процесса, на котором должен быть получен результат
- **comm** – коммуникатор, внутри которого выполняется операция
- **request** – дескриптор операции

Неблокирующие коллективные обмены… Распределение и сбор данных

- Чтобы распределить данные с выбранного процесса по всем процессам

```
int MPI_Iscatter(void *sbuf, int scount, MPI_Datatype stype,
  void *rbuf, int rcount, MPI_Datatype rtype,
  int root, MPI_Comm comm, MPI_Request *request);
```

- **sbuf, scount, stype –** параметры передаваемого сообщения (**scount** определяет число элементов,передаваемых каждому процессу)
- **rbuf, rcount, rtype –** параметры получаемого сообщения
- **root –** номер процесса, на котором должен быть получен результат
- **comm –** коммуникатор, внутри которого выполняется операция
- **request –** дескриптор операции

- Если размеры сообщений для процессов различны, распределение данных проводится с помощью функции **MPI_Iscatterv()**

Неблокирующие коллективные обмены… Распределение и сбор данных

- Сбор данных всех процессов для одного процесса – операция, противоположная разделению данных

```
int MPI_Igather(void *sbuf, int scount, MPI_Datatype stype,
  void *rbuf, int rcount, MPI_Datatype rtype,
  int root, MPI_Comm comm, MPI_Request *request);
```

- **sbuf, scount, stype** – параметры передаваемого сообщения
- **rbuf, rcount, rtype** – параметры получаемого сообщения
- **root** – номер процесса, на котором должен быть получен результат
- **comm** – коммуникатор, внутри которого выполняется операция
- **request** – дескриптор операции

- Если размеры сообщений для процессов различны, сбор данных проводится с помощью функции **MPI_Igatherv()**

Неблокирующие коллективные обмены… Всеобщие обмены

- Чтобы получить все собранные данные на каждом процессе коммуникатора, необходимо использовать функцию сбора и рассылки **MPI_Iallgather()**

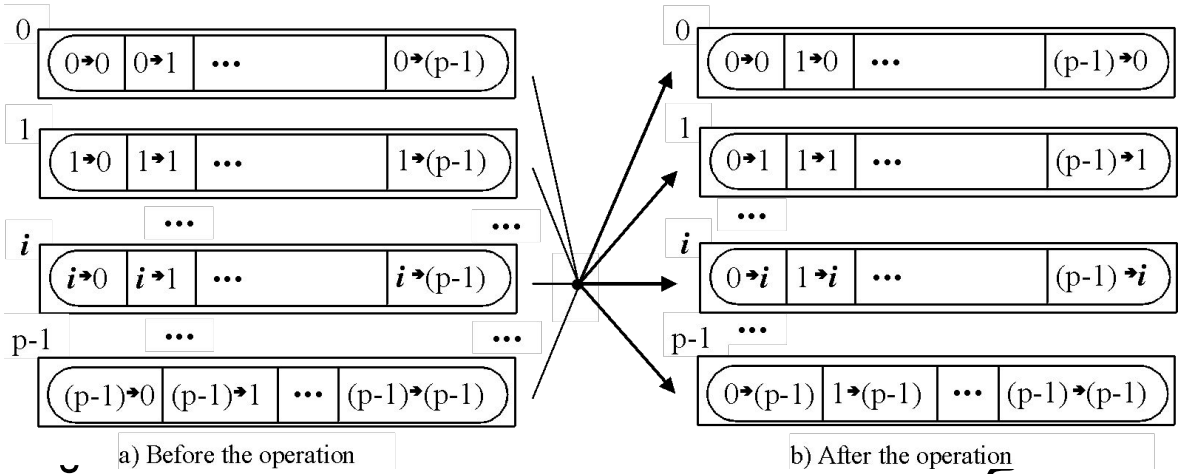
```
int MPI_Iallgather(
  void *sbuf, int scount, MPI_Datatype stype,
  void *rbuf, int rcount, MPI_Datatype rtype,
  MPI_Comm comm, MPI_Request *request);
```

- Исполнение общего варианта операции сбора данных, когда размеры сообщений, передаваемых между процессами, могут различаться, проводится с помощью функции **MPI_Iallgatherv()**

Неблокирующие коллективные обмены… Всеобщие обмены

- Общий обмен данных между процессами

```
int MPI_Ialltoall(void *sbuf, int scount, MPI_Datatype stype,
  void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm,
  MPI_Request *request);
```



- Вариант этой операции, когда размеры сообщений, передаваемых между процессами, могут различаться, проводится с помощью функции **`MPI_Ialltoallv()`**

Неблокирующие коллективные обмены… Всеобщие обмены

- Чтобы получить результаты редукции данных на каждом процессе коммуникатора, необходимо использовать функцию **`MPI_Iallreduce()`**

```
int MPI_Iallreduce(void *sendbuf, void *recvbuf, int count,
  MPI_Datatype type, MPI_Op Op, MPI_Comm comm,
  MPI_Request *request);
```

- Для операций, не являющихся ассоциативными, результат, полученный при завершении неблокирующей редукции (через **`MPI_Ireduce()`** or **`MPI_Iallreduce()`**), **может быть неточно равным** результату, полученному блокирующей редукцией

Неблокирующие коллективные обмены
Синхронизация вычислений

- **MPI_IBarrier()** - неблокирующая версия **MPI_Barrier()**

```
int MPI_Ibarrier(MPI_Comm comm, MPI_Request *request);
```

- Вызывая **MPI_IBarrier()**, процесс объявляет, что он достиг барьера
- Вызов возвращается немедленно, независимо от того, вызывали ли другие процессы **MPI_IBarrier()**
- Неблокирующий барьер может быть использован для скрытия задержки
- Перемещение независимых вычислений между **MPI_IBarrier()** и последующим завершением вызова (**MPI_Wait()**, …) может перекрыть задержку барьера

Создание и управление процессами

- Общее описание
- Динамическая модель процессов
- Управление процессами
- Установка соединений

Создание и управление процессами
Общее описание

- Важные классы приложений MPI, которым требуется контроль над процессами
  - ферма задач
  - последовательно запускаемые приложения с параллельными модулями
  - проблемы, требующие проведения оценки числа и типов процессов, которые должны быть запущены, во время выполнения
- MPI обеспечивает ясный интерфейс между приложением и системным программным обеспечением
- MPI гарантирует предсказуемость обменов при присутствии динамических процессов
- MPI поддерживает сохранение концепции коммуникатора независимо от того, как появились его элементы
- Коммуникатор никогда не изменяется после создания, и он всегда создается с помощью определенных коллективных операций

Создание и управление процессами… Динамическая модель процессов

- Динамическая модель процессов делает возможным создание и совместное завершение процессов после запуска приложения MPI
- Она предоставляет механизм для установки связи между недавно созданными процессами и существующими приложениями MPI
- Она также предоставляет механизм для установки связи между двумя существующими приложениями MPI, даже если ни одно из них не участвовало в запуске другого

Создание и управление процессами… Динамическая модель процессов

Запуск процессов

- Приложения MPI могут запускать новые процессы с помощью интерфейса к внешнему менеджеру процессов
  - **`MPI_Comm_spawn()`** запускает процессы MPI и устанавливает с ними соединение, возвращая интеркоммуникатор
  - **`MPI_Comm_spawn_multiple()`** запускает несколько разных исполняемых файлов (или тот же исполняемый файл с разными аргументами), помещая их в одном **MPI_COMM_WORLD** и возвращая интеркоммуникатор
- Процесс представлен в MPI парой (группа, номер)
- Пара (группа, номер) представляет уникальный процесс
- Процесс может обладать несколькими парами (группа, номер), ведь процесс может принадлежать нескольким группам

Создание и управление процессами…
Управление процессами

```
int MPI_Comm_spawn(const char *command, char *argv[],
  int maxprocs, MPI_Info info, int root, MPI_Comm comm,
  MPI_Comm *intercomm, int array_of_errcodes[]);
```

- **MPI_Comm_spawn()** пытается запустить **maxprocs** идентичных копий программы MPI, указанной в **command**, устанавливая с ними соединение и возвращая интеркоммуникатор
- Запущенные процессы считаются потомками. У потомков есть свой **MPI_COMM_WORLD**, отличный от родительского
- **MPI_Comm_spawn()** коллективный для **comm**
- Интеркоммуникатор, возвращенный **MPI_Comm_spawn()**, содержит процессы-предки в локальной группе, а процессы-потомки - в удаленной группе

Создание и управление процессами…
Управление процессами

```
int MPI_Comm_spawn(const char *command, char *argv[],
  int maxprocs, MPI_Info info, int root, MPI_Comm comm,
  MPI_Comm *intercomm, int array_of_errcodes[]);
```

- **command** - имя запускаемой программы (значимо только для корня)
- **argv** - аргументы команды (значимо только для корня)
- **maxprocs** - максимальное число запускаемых процессов (значимо только для корня)
- **info** - набор пар ключ-значение, передающий исполняемой системе, где и как запускать процессы (значимо только для корня)
- **root** - номер процесса, в котором рассматриваются предыдущие аргументы (корневого процесса)
- **comm** - интеркоммуникатор, содержащий группу запускаемых процессов
- **intercomm** - интеркоммуникатор между первичной и недавно запущенной группами
- **array_of_errcodes** - один код на один процесс (массив integer)

Создание и управление процессами…
Управление процессами

```
int MPI_Comm_spawn_multiple(int count, char *commands[],
  char **argvs[], const int maxprocs[],
  const MPI_Info info[], int root, MPI_Comm comm,
  MPI_Comm *intercomm, int array_of_errcodes[]);
```

- **MPI_Comm_spawn_multiple()** идентичен **MPI_Comm_spawn()** за исключением того, что здесь указывается несколько исполняемых файлов
- Первый аргумент, **count**, определяет их число
- Каждый из следующих 4 аргументов - массивы соответствующих аргументов из **MPI_Comm_spawn()**
- Все запущенные процессы обладают тем же **MPI_COMM_WORLD**

Создание и управление процессами…
Управление процессами

```
int MPI_Comm_spawn_multiple(int count, char *commands[],
  char **argvs[], const int maxprocs[],
  const MPI_Info infos[], int root, MPI_Comm comm,
  MPI_Comm *intercomm, int array_of_errcodes[]);
```

- **count** – число команд
- **commands** – программы к исполнению
- **argvs** – аргументы для команд
- **maxprocs** – макс. число запускаемых процессов для каждой команды
- **infos** – информационные объекты, передающий исполняемой системе, где и как запускать процессы
- **root** – номер процесса, в котором рассматриваются предыдущие аргументы
- **comm** – интеркоммуникатор, содержащий группу запускаемых процессов
- **intercomm** – интеркоммуникатор между первичной и недавно запущенной группами
- **array_of_errcodes** – один код на один процесс (массив integer)

Создание и управление процессами… Установка соединений

- Несколько ситуаций, когда установка соединений полезна:
  - Двум частям приложения, запущенным независимо друг от друга, необходимо произвести обмен
  - Инструмент визуализации хочет присоединиться к работающему процессу
  - Сервер хочет принимать соединения от нескольких клиентов. Как клиенты, так и сервер могут быть параллельными программами
- MPI должен установить каналы обмена там, где нет связи предка/потомка.

Создание и управление процессами… Установка соединений

- MPI должен установить каналы обмена там, где нет отношений предок/потомок
  - Установка связи между двумя группами процессов, не связанными существующим коммуникатором - коллективный, но асимметричный процесс
  - Одна группа процессов показывает желание получить соединение с другими группами процессов
  - Назовем эту группу сервером, даже если это не приложение типа клиент/сервер
  - Другая группа подключается к серверу; назовем ее клиентом

Создание и управление процессами…
Установка соединений

Команды сервера

```
int MPI_Open_port(MPI_Info info, char *port_name);
```

- Устанавливает сетевой адрес, зашифрованный в строке **port_name**, по которому сервер сможет принимать соединения с клиентами

```
int MPI_Close_port(const char *port_name);
```

- Освобождает сетевой адрес, представленный в **port_name**

```
int MPI_Comm_accept(const char *port_name, MPI_Info info,
  int root, MPI_Comm comm, MPI_Comm *newcomm);
```

- Устанавливает соединение с клиентом
- Вызов коммуникатора совершается коллективно. Он возвращает интеркоммуникатор, разрешающий обмен с клиентом

Создание и управление процессами…
Установка соединений

Команды клиента

```
int MPI_Comm_connect(const char *port_name, MPI_Info info,
  int root, MPI_Comm comm, MPI_Comm *newcomm);
```

- Устанавливает обмен с сервером, указанным в **`port_name`**.
- Вызов коммуникатора совершается коллективно и возвращает интеркоммуникатор, в который удаленная группа поступила вызовом **`MPI_Comm_accept()`**
- Если указанного порта не существует (или он был закрыт), **`MPI_Comm_connect()`** возвращает ошибку класса **`MPI_ERR_PORT`**

Создание и управление процессами
Установка соединений

Команды клиента

```
int MPI_Comm_connect(const char *port_name, MPI_Info info,
  int root, MPI_Comm comm, MPI_Comm *newcomm);
```

- Если порт существует, но у него нет незавершенных **MPI_Comm_accept()**, попытка соединения рано или поздно прервется после интервала времени, зависящего от реализации, или удачно завершится, если сервер вызовет **MPI_Comm_accept()**
- В случае, если время соединения истекло, **MPI_Comm_connect()** возвращает ошибку класса **MPI_ERR_PORT**

Заключение

- Обсудили неблокирующие коллективные операции
- Создание и управление дополнительными процессами в программе MPI рассмотрены

Упражнения

- Разработайте тестовую программу для каждого метода неблокирующих коллективных операций
- Разработайте тестовую программу, используя дополнительные процессы в программе MPI. Возможная схема реализации - “Мастер - рабочие”

Литература

1. Интернет-ресурс, описывающий стандартный MPI: http://www.mpiforum.org
2. Одна из наиболее широко используемых реализаций MPI, библиотека MPICH, представлена на сайте http://www.mpich.org
3. Quinn, M.J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
4. Pacheco, P. (1996). Parallel Programming with MPI. - Morgan Kaufmann.
5. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. (1996). MPI: The Complete Reference. – MIT Press, Boston, 1996.
6. Group, W., Lusk, E., Skjellum, A. (1999). Using MPI – 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). – MIT Press.
7. Group, W., Lusk, E., Thakur, R. (1999). Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation). – MIT Press.