Многопоточные вычисления на основе технологий MPI и OpenMP: MPI. Введение

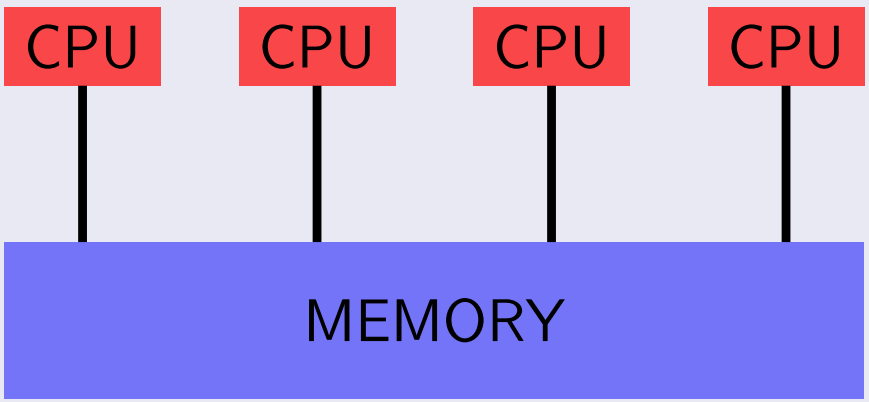
Н. И. Хохлов

МФТИ, Долгопрудный

14 сентября 2016 г.

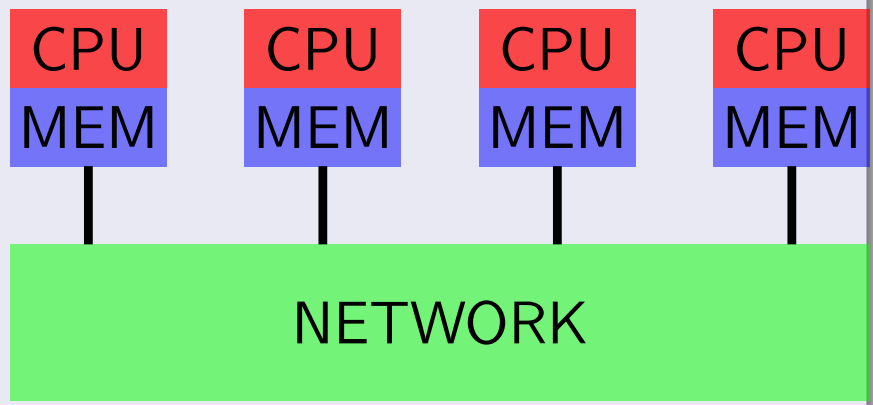
Архитектуры параллельных вычислительных машин

Общая память (shared memory)



Пример – современные много-ядерные/многопроцессорные машины.

Распределенная память (distributed memory)



Пример – несколько одноядерных машин объеденeнных в сеть.

Большинство современных архитектур – гибридные, т. е. многоядерные машины, объеденeнные в одну общую сеть.

Что такое MPI?

- MPI = Message Passing Interface – Интерфейс для передачи сообщений.
- MPI – не библиотека, а спецификация (стандарт) для программистов и пользователей. На основе спецификации может быть написана библиотека.
- Основная цель MPI – предоставить широко используемый стандарт для написания параллельных приложений построенных на передаче сообщений.
- Интерфейс есть для языков C и Fortran. Некоторые версии стандарта также поддерживают C++.

История появления стандарта MPI

- 1980-1990 гг. – появление суперкомпьютеров с разделяемой памятью.
- 1992-1994 гг. – множество технологий для написания приложений в системах с разделяемой памятью, начало зарождения MPI.
- Апрель 1992 г. – начата работа над спецификацией MPI, были обсуждены основные идеи и функциональность. Далее шла работа над спецификацией (Center for Research on Parallel Computing, Williamsburg, Virginia).
- Ноябрь 1992 г. – встреча в Минеаполе (Minneapolis). Черновой вариант MPI. Создание MPI Forum (MPIF) – туда входит около 175 членов из 40 организация занимающихся параллельными вычислениями, программный обеспечением а также академические и научные организации.

История появления стандарта MPI

- Ноябрь 1993 г. – на конференции Supercomputing 93 были доложены стандарты MPI.
- Май 1994 г. – финальная версия стандарта MPI – http://www-unix.mcs.anl.gov/mpi.
- 1996 г. – появление спецификации MPI-2, прежняя спецификация получила название MPI-1.
- Сентябрь 2012 г. – стандарт MPI-3.

Современные реализации MPI включают в себя MPI-1, MPI-2, MPI-3, в зависимости от реализации возможна поддержка либо только части стандартов или все вместе.

Почему надо использовать MPI?

- **Стандарт** – единственный стандарт HPC на текущий момент.
- **Переносимость кода** – нет необходимости менять код при использовании различных платформ.
- **Производительность** – производители железа и софта сами заботятся о скорости работы библиотек.
- **Функциональность** – только стандарт MPI-1.1 предоставляет более 115 функций.
- **Доступность** – множество свободных реализаций.
- **Простота отладки** – в отличии от приложений на системах с общей памятью, каждый MPI процесс работает однопоточно (в рамках MPI).

Идеология MPI

Передача сообщений совместная операция, она возникает только когда один процесс *отсылает* сообщение, а второй *принимает*.

Отправитель

- Должен знать что отправляет и какого размера (указанель на область локальной памяти и размер).
- Кому отправляет. В простейшем случае адрессации это может быть число (номер процесса).

Получатель

- Данные необходимо принять, соответственно указатель на локальную память и ее размер.
- Указатель на отправителя, соответственно его номер.
- Опционально может принять меньше данных, чем указал.

Идеология MPI

Процесс должен иметь механизм для контроля сообщений, которые к нему приходят. Он может отсекать или различать сообщения от другого процесса. Также процесс отправитель может сортировать или както различать сообщения, которые отсылает.

Вводится понятие типа - *type* или тега сообщения - *tag*. Используется понятие *tag*, поскольку тип уже много где занят.

Example (Возникает интерфейс)

send(address, length, destination, tag)
и
receive(address, length, source, tag, actlen)

Входной буффер

Задание данных сообщения вида *(address, length)* не всегда удобно:

- Часто данные расположены в памяти непоследовательно.
- От типа данных (целое число, строка, число с плавающей точкой и т.д.) зависит то, как они лежат в памяти на различных системах.

Возникает понятие типа данных:

Example (тип данных)

(address, count, datatype)

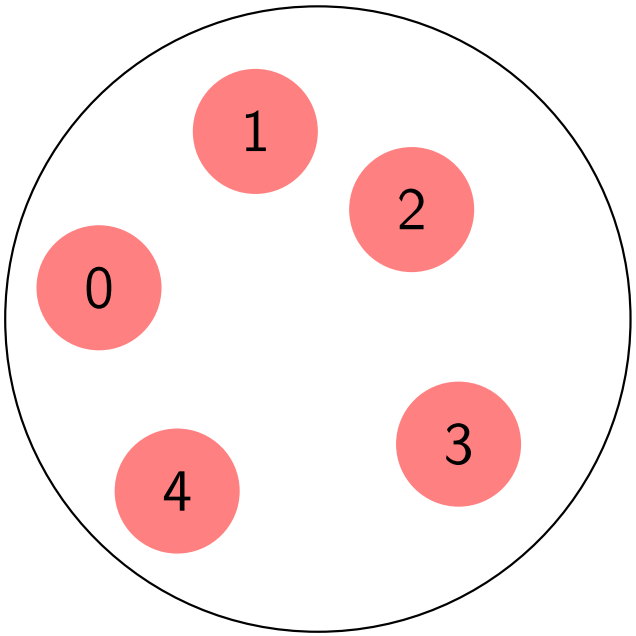
- Идеология представления данных в виде массива, по адресу *address* лежит *count* данных типа *datatype*.
- Типы данных могут описывать данные, не лежащие последовательно в памяти.

Разделение сообщений

Первые системы построенные на передаче сообщений требовати жесткого задания tag у сообщений на стороне отправители и принимателе. Пользователю требуется самому оперировать заданием tag. Можно использовать wild-card прием, используя специальное значение принимать сообщение с любым tag.

В MPI дополнительно вводится понятие *контекста*. Контекст создается автоматически при запуске приложения и может дополнительно выделятся по запросу пользователя. Позволяет дополнительно регулировать сообщения между процессами. (Реализовано через коммуникаторы).

Нумерация процессов



- Процессы состоят в *группах*.
- Внутри группы каждый процесс имеет уникальный нормер (rank, task id).
- Номер представляет собой целое, неотрицательное число. Номера идут последовательно от 0 с шагом 1.
- Если размер группы N, то номера процессов лежат в интервале 0...N-1.
- Удобно использовать для контроля выполнения программы на разных процессах (if (rank == 0)).

Комуникатор

Понятие группы и контекста объединяются в *коммуникаторе*.

Требуется в качестве аргумента во всех операциях взаимодействия.

Example (Новый интерфейс для сообщений в MPI)

MPI_Send(address, count, datatype, destination, tag, comm)
и
MPI_Recv(address, maxcount, datatype, source, tag, comm, status)

MPI_Send(address, count, datatype, destination, tag, comm)

- *(address, count, datatype)* – описывает count объектов типа datatype, лежащих по адресу address.
- *destination* – номер процессора получатеся в группе соответствующей коммуникатору comm.
- *tag* – tag сообщений (целое число) для разделения сообщений.
- *comm* – коммуникатор, описывающий контекст взаимодействия.

MPI_Recv(address, maxcount, datatype, source, tag, comm, status)

- *(address, maxcount, datatype)* – описывает буфер, который может принять максимум maxcount объектов типа datatype, лежащих по адресу address.
- *source* – номер процессора отправителя в группе соответствующей коммуникатору comm, возможно принятие от любого процесса в группе (wild-card).
- *tag* – tag сообщений (целое число) для разделения сообщений, возможен прием с любым тегом (wild-card).
- *comm* – коммуникатор, описывающий контекст взаимодействия.
- *status* – статус сообщения, хранящий фактический размер принятого сообщения, номер процесса отправителя и тег.

Дополнительный функционал MPI-1

- *Коллективные операции*. Дополнительный функционал для передачи данных между несколькими процессами или коллективные вычисления.
- *Виртуальные топологии*. Можно создать топологию процессов, которая наилучшим образом ложиться под конкретную задачу. Поддерживает топологии в виде декартовой сети и графа.
- *Отладка и профилировка*. Дает ряд функционала по отладке и профилировке приложений.
- *Режимы взаимодействия процессов*. Синхронные и асинхронные режимы взаимодействия.

Модель программирования MPI

- Дает виртуальный интерфейс ко всем моделям программирования с распределенной памятью.
- Железо:
  - Компьютеры с распределенной памятью – изначально разрабатывалась для них.
  - Общая память – дает виртуальную распределенную память.
  - Гибридные – современные версии MPI дают большие возможности для работы на гибридных архитектурах, в том числе с наличием GPU процессоров.
- Явный параллелизм.
- Число процессов статично. Нельзя породить новый процесс во время работы программы (MPI-2 и далее обходит это ограничение).
- MPI процесс – обычный процесс операционной системы.

Стиль кода C/C++

- Объявление всех функций и типов данных находится в заголовосном файле *mpi.h*.
- Функция, тип данных, константа начинается с префикса *MPI_*.
- Константы идут заглавными буквами. Например: *MPI_INT*, *MPI_COMM_WORLD* и т.д.
- Названия функций идут с первой заглавной буквы, затем строчные. Например: *MPI_Comm_size*, *MPI_Type_vector* и т.д.
- Типы данных называются аналогично функциям. Например: *MPI_Comm*, *MPI_User_function* и т.д.

Пример "Hello, world!"

```
#include <stdio.h>


int main(int argc, char *argv[])
{




    printf("Hello, world!\n");

    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{




    printf("Hello, world!\n");

    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{

    MPI_Init(&argc, &argv); // Инициализация MPI.


    printf("Hello, world!\n");

    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{

    MPI_Init(&argc, &argv); // Инициализация MPI.


    printf("Hello, world!\n");
    MPI_Finalize(); // Завершение работы с MPI.
    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{
    int  numtasks, rank; // Номер и число процессов.
    MPI_Init(&argc, &argv); // Инициализация MPI.


    printf("Hello, world!\n");
    MPI_Finalize(); // Завершение работы с MPI.
    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{
    int  numtasks, rank; // Номер и число процессов.
    MPI_Init(&argc, &argv); // Инициализация MPI.
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks); // Число потоков.

    printf("Hello, world!\n");
    MPI_Finalize(); // Завершение работы с MPI.
    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{
    int  numtasks, rank; // Номер и число процессов.
    MPI_Init(&argc, &argv); // Инициализация MPI.
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks); // Число потоков.
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Номер текущего пот
    printf("Hello, world!\n");
    MPI_Finalize(); // Завершение работы с MPI.
    return 0;
}
```

Пример "Hello, world!"

```
#include <stdio.h>
#include <mpi.h> // Заголовочный файл MPI.

int main(int argc, char *argv[])
{
    int  numtasks, rank; // Номер и число процессов.
    MPI_Init(&argc, &argv); // Инициализация MPI.
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks); // Число потоков.
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Номер текущего пот
    printf("Number of tasks= %d My rank= %d\n  numtasks, rank);
    MPI_Finalize(); // Завершение работы с MPI.
    return 0;
}
```

- MPI_Init(int *argc, char ***argv) – инициализация MPI-окружения. Все функции взаимодействия должны вызываться только после данной функции.
  - argc – указатель на параметр argc функции main.
  - argv – указатель на параметр argv функции main.
- MPI_Finalize() – завершение работы с MPI.
- MPI_Comm_size(MPI_Comm comm, int *size) – узнать число процессов в коммуникаторе (размер коммуникатора).
  - comm – коммуникатор MPI.
  - size – указатель на переменную, куда будет записано число процессов.
- MPI_Comm_rank(MPI_Comm comm, int *rank) – узнать номер данного процесса в коммуникаторе (rank).
  - comm – коммуникатор MPI.
  - rank – указатель на переменную, куда будет записан номер процесса.

Компиляция

- Предоставляет свою обертку для стандартного компилятора в системе.
- Названия компиляторов в среде Linux: mpicc, mpiCC, mpicxx, mpic++.
- Вызывает сторонний компилятор в системе (gcc, icc и т. д.) с набором опций.
- Для работы библиотеки необходимо подключить заголовочный файл mpi.h.

Example (Компиляция)

```
mpicc -o hello hello.c
```

Запуск

- Для инициализации окружения MPI необходим запуск через ее программы.
- Параметры запуска сильно зависят от версии библиотеки и используемого окружения.
- Для запуска на нескольких узлах использует rsh/ssh протокол.
- Основная команда для запуска – mpirun. Число процессов задается опцией -np.

Example (запуск)

```
mpirun -np 5 ./hello
```

Запуск

```
user@host:~/$ mpicc -o hello mpi_hello.c
user@host:~/$ mpirun -np 5 ./hello
Number of tasks = 5 My rank = 0
Number of tasks = 5 My rank = 3
Number of tasks = 5 My rank = 1
Number of tasks = 5 My rank = 4
Number of tasks = 5 My rank = 2
user@host:~/$
```

Особенности запуска

- Порядок строк может быть произвольным.
- MPI окружение гарантирует, что символы в отдельных строках различных процессов не будут перемешиваться.
- Запуск без команды mpirun обычно приводит к работе приложения в один поток.

Доработка

Доработать пример, чтобы процессы обменивались сообщениями. Реализовать кольцевую пересылку номера процесса соседу справа, полученное значение процесс выводит на экран. Последний процесс отсылает значение нулевому процессу.

Спасибо за внимание! Вопросы?