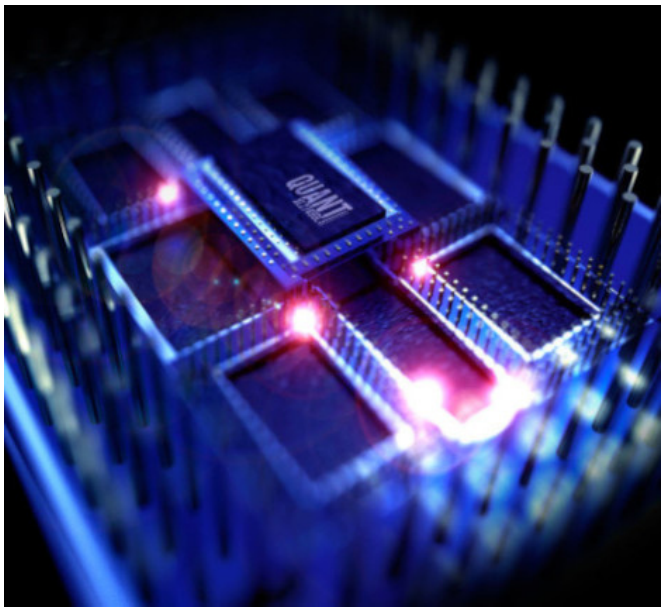
Введение в MPI – 1



П.А. Тараканов

Кафедра астрофизики
математико-механического факультета
Санкт-Петербургского государственного
университета

5 апреля 2022 г.

Общие сведения

- MPI — стандартная библиотека для работы в системах с *распределенной* памятью (суперкомпьютеры, Beowulf кластера).
- Идеологически написание программ под MPI похоже на OpenMP, но отсутствие единой памяти означает, что каждый процесс должен хранить свои данные самостоятельно, обмен ими — «дорогое» действие.
- Идеология MPI — обмен *сообщениями*: информацией о состоянии, командами или блоками данных.

Как компилировать и запускать программы

- Для сборки используются `mpif90` (Fortran 90+), `mpif77` (Fortran 77), `mpicc` (C), `mpiCC` (C++). Это «обертки» к компиляторам семейства GCC, так что прочие параметры такие же, как там.
- Запуск программ — командой `mpiexec` (или `mpirun`, как правило, разницы нет).
- Минимальный набор параметров при запуске:
  - `-n <число>` (или `-np <число>`) — задание *суммарного* числа потоков на всех узлах кластера;
  - `-N <число>` — задание числа потоков на *каждом* узле кластера;
  - `-host <список имен>` — задание списка узлов (через запятую, без пробелов, после имени `:<число>` — число ядер на узле);
  - `-hostfile <имя файла>` — задание файла, содержащего список узлов. Каждая строка в файле имеет вид
    `<имя узла> [slots=<число> [max_slots=<число>]]`
    (в квадратных скобках необязательные параметры, указывающие штатное число потоков на узле и максимально возможное их число).

Как работать с местным кластером

- Конфигурация суперкомпьютеров/кластеров почти всегда делает их уникальными, так что необходимо интересоваться местными особенностями.
- Тут установлена OpenMPI 4.0.
- Доступные машины: от `stud01` до `stud17` (`08` не работает).
- Нужен беспарольный вход по SSH: запускаете `ssh-keygen` и на все соглашаетесь, после чего запускаете `ssh-copy-id studserver` и вводите свой пароль. После этого ssh-переходы между машинами должны работать без пароля.
- На всех компьютерах по 4 ядра.
- Если при запуске система будет ругаться на «медленный транспорт», можно запускать программы с дополнительным параметром `--mca btl tcp`.

Задача номер 0

- Настройте себе беспарольный SSH на машинах класса (если не делали этого ранее).
- В каталоге

  `/common/Учебные материалы/Программирование/MWIPS`

  лежат исходники теста производительности Whetstone (классического синтетического теста производительности вычислительных систем на задачах арифметики с плавающей точкой).
- Скопируйте их оттуда, соберите и запустите на нескольких узлах (4–6, больше не надо, оставьте что-нибудь соседям).

Основные детали MPI-программы

- Для использования стандартных процедур и функций нужно сделать `use :: mpi` (Fortran 90+), `include 'mpif.h'` (Fortran 77), `#include<mpi.h>` (C/C++).
- В начале программы с использованием MPI надо вызвать `call MPI_Init(Err)`, в программе это делается только один раз.
- В конце программы с использованием MPI надо вызвать `call MPI_Finalize(Err)`.
- `Err` — переменная типа `integer(4)`, содержит код ошибки (или `MPI_SUCCESS`, если все в порядке).
- Делать что-то за пределами `MPI_init` и `MPI_finalize` *не надо*, последствия труднопредсказуемы.
- В C/C++ все аналогично, но там MPI во всех названиях обязательно должны быть большими, первая буква оставшейся части названия — тоже, остальные буквы в названиях функций — строчные, а в названиях констант — заглавные. Во всех случаях функции C/C++ возвращают целое (код ошибки Err).

Коммуникаторы

- В MPI процессы относятся к *коммуникаторам* (группам процессов).
- Стандартный коммуникатор `MPI_COMM_WORLD` — включает все процессы сразу.
- Можно создавать дополнительные коммуникаторы, один процесс может быть связан с несколькими коммуникаторами.
- Тип коммуникатора — целый (в C есть тип `MPI_Comm`).
- `call MPI_Comm_size(MPI_COMM_WORLD, Size, Err)` — узнать размер коммуникатора (количество процессов в нем), можно подставить и другой коммуникатор.
- `call MPI_Comm_rank(MPI_COMM_WORLD, Rank, Err)` — узнать ранг процесса (номер в коммуникаторе, нумерация начинается с нуля).

Служебные функции

- `MPI_Wtime(Err)` — *функция* (и в Фортране тоже), возвращающая UNIX-время в секундах (в `real(8)`).
- `MPI_Wtick(Err)` — также *функция*, возвращающая разрешение таймера в секундах.
- `MPI_Get_processor_name(Name,Len,Err)` — процедура вовращает имя узла для процесса, длину имени и код ошибки.

Типы сообщений

- Процессы могут обмениваться сообщениями, причем бывают разные варианты подтипов сообщений.
- По набору участников:
  - Передача “point-to-point” (PtP): участвуют два процесса, один отправляет сообщение, другой получает. Должен иметься общий коммуниктор. Получатель может не знать номер отправителя, отправитель всегда должен знать номер получателя.
  - Передача всем процессам коммуникатора: один процесс сообщает что-то всем остальным в коммуникаторе (включая себя).
- По блокировке:
  - Передача с блокировкой, выполнение процесса останавливается до окончания передачи.
  - Асинхронная передача, процесс продолжает свою деятельность сразу после отправки сообщения.

Передача PtP-сообщения с блокировкой

- Для передачи блокирующего сообщения используется процедура `MPI_Send(Buf, Count, Datatype, Dest, MsgTag, Comm, Err)`.
- `Buf` — массив из `Count` элементов типа `Datatype`.
- Все остальные параметры, кроме `Buf` — целые.
- `Count` в принципе может быть равен 0 (если нужен только сам факт сообщения, но не данные).
- Есть стандартный набор констант для типов, например, для `real(8)` это `MPI_REAL8`, а также `MPI_BYTE` для нетипизированных и `MPI_PACKED` для упакованных данных.
- `Dest` — номер процесса, которому посылается сообщение. Можно посылать самому себе, но для блокирующих сообщений это чревато неприятностями.
- `MsgTag` — идентификатор сообщения.
- `Comm` — коммуникатор, в рамках которого посылаем.

Передача PtP-сообщения с блокировкой

- Есть (вернее, нет) стандартный процесс `MPI_PROC_NULL` — несуществующий процесс, посылка любых сообщений ему всегда оканчивается успешно.
- После отработки посылки можно менять содержимое посылаемых данных, но нет гарантии того, что данные переданы получателю. Для более аккуратной обработки подобных ситуаций есть варианты процедуры:
  - `MPI_bsend` — передача с буферизацией. Данные записываются в специальный буфер и там ждут приема. Нужно создать буфер вызовом `call MPI_Buffer_attach(Buf, Size, Err)` — нужно завести массив `Buf` размера `Size`. Освобождение обратно: `call MPI_Buffer_detach(Buf, Size, Err)`. Тип массива неважен, лучше `byte`.
  - `MPI_ssend` — передача с синхронизацией. Процедура закончится, когда получатель начнет прием.
  - `MPI_rsend` — передача при условии, что получатель уже готов к приему. Проблематично, но быстрее, чем предыдущие.

Прием сообщения

- Для приема сообщения используется процедура `MPI_Recv(Buf, Count, Datatype, Source, MsgTag, Comm, Status,Err)`.
- Смысл параметров в основном тот же, что и при передаче, `Count` и `Datatype` надо задавать.
- `Source` — номер того, от кого принимаем сообщение.
- `Status` — целочисленный массив размера `MPI_STATUS_SIZE`, в котором есть такие элементы:
  - `Status(MPI_SOURCE)` — номер процесса-отправителя;
  - `Status(MPI_TAG)` — идентификатор сообщения;
  - `Status(MPI_ERROR)` — код ошибки.

  (в C/C++ это структура типа `MPI_Status` с аналогичными названиями полей).
- Все это нужно, поскольку в качестве источника можно использовать `MPI_ANY_SOURCE` — принимать сообщения от любого процесса, а в качестве идентификатора `MPI_ANY_TAG` — принимать сообщения с любым идентификатором.

Предварительная информация о принимаемом

- Есть процедура `MPI_Probe(Source, Msgtag, Comm, Status, Err)`, позволяющая «попробовать» принять сообщение. Полученную информацию можно обработать, а потом принять то же сообщение с помощью `MPI_recv`.
- Можно также воспользоваться процедурой `MPI_Get_count(Status, Datatype, Count, Err)`. По уже известному `Status` определяется тип и количество элементов, которые придут в сообщении — полезно для организации массива под их хранение.

Задачи

- Выполнить задачу 0 (см. выше).
- Попытаться написать произвольную программу, в которой процессы будут обмениваться сообщениями и в результате выводить что-либо на экран.