Введение в MPI

Комментарии к слайдам
Автор: Алексей А. Романенко
arom@ccfit.nsu.ru

Слайд1, 2

В разделе рассматривается библиотека для реализации программ на системах с распределенной памятью. Из раздела вы узнаете как появился MPI, что это такое и как с его помощью писать, компилировать и запускать программы.

Слайд 3
Прежде чем переходить к MPI давайте вспомним что такое система с распределенной памятью. Под распределенной памятью подразумевают оперативную память. Такие системы состоят из компьютеров или узлов, которые объединены в сеть. Сеть может быть как FastEthternet, GigabitEthernet, так и специализированные для таких вычислительных систем решения, напримтр, Infiniband или Myrinet. В упрощенном варианте процессор к своей памяти (в своем узле) может доступаться напрямую, а к памяти другого узла только через специальные запросы. Для выполнения тяжелых расчетов собираются специализированные системы, где как правило, все узлы имеют одинаковую производительность, весь комплекс располагается в одном помещение и этот комплекс называется кластером. Конечно можно собрать кластер и из компьютеров терминального класса или общежития, но у него не будет той стабильности и надежности, которая предъявляется к подобным системам.

Слайд 4
Если кому-то доводилось писать клиент-серверные приложения, то он вспомнит необходимые действия, которые надо проделать, чтобы передать информацию по сети. На слайде представлена последовательность действий для серверной (слева) и клиентской части (справа) программы. И это только между двумя процессами. На кластере запускаются задачи состоящие из десятков подпроцессов взаимодействующих между собой и писать для каждого из них приведенный псевдокод утомительно и отвлекает от основной задачи.

Слайд 5
Чтобы избавить людей, занимающихся расчетами, от этих и других забот, связанных с передачей данных по сети, были разработаны специализированные библиотеки. Они представлены на слайде. PVM - параллельная виртуальная машина и MPI — интерфейс передачи сообщений. Отличие их (помимо набора функций) в том, что MPI — библиотека, которая непосредственно реализует передачу данных по сети и на стадии сборки линкуется с программой, а у PVM на узле запускается процесс-демон, который ответственен за передачу сообщений между узлами. MPI сейчас является стандартом и установлен на всех кластерах.

Слайд 6
Рождением MPI можно считать июнь 1994 года, когда после длительных дискуссий на специальном форуме был представлен документ, описывающий набор функций, правило их наименования и ряд других моментов, которые необходимы для написания параллельных программ для кластеров.

Слайд 7
Спустя три года стандарт MPI был расширен и дополнен с учетом пожеланий пользователей. Так появился MPI-2.

Слайд 8
Таким образом MPI - является спецификацией на библиотечные функции. В MPI-2 добавлен параллельный В/В динамическое управление процессами, удаленные операции в памяти и пр. Подробно ознакомиться со стандартом можно по представленным ссылкам.
Слайд 9
При разработке MPI преследовались следующие цели: обеспечить механизм передачи и сообщений и избавить разработчиков от рутинной работы; обеспечить переносимость кода между кластерами; обеспечить эффективную реализацию некоторых операций по передаче данных. Таким образом MPI предлагает большой объем функциональности (около сотни различных функций), поддержку гетерогенных архитектур.

Слайд 10
Для того, чтобы была возможность использовать MPI функции, необходимо подключить заголовочный файл: mpi.h для C/C++ и mpif.h для Фортрана. Компилирование программы выполняется компилятором с подключением соответствующей библиотеки или можно вызвать скрипт, который подставит все требуемые параметры за вас. Запуск программы производится с помощью команды mpirun и через ключ -np указывается количество подпрограмм, которое требуется запустить.

Слайд 11
При запуске каждый процессор выполняет одну и туже программу, т.е. по классификации Флинна это SPMD подход. Все процессы имеют одинаковые по названию переменные, но значение этих переменных может отличаться. После запуска подпроцессы взаимодействуют через обмен сообщениями.

Слайд 12
В созданной группе процессов каждый процесс идентифицируется rank-ом — целым числом от нуля до количества процессов в группе минус один. Распределение данных идет на основе значения rank.

Слайд 13
Не смотря на то, что MPI ориентирован на стиль SPMD при необходимости можно реализовать и MPMD вариант. Для этого не только данные но и исполняемые функции должны зависеть от ранка программы.

Слайд 14
Вариант реализации MIMD подхода.

Слайд 15
Из названия MPI понятно, что коммуникация происходит через передачу сообщений между подпрограммами. Для передачи сообщения необходимо указать следующие атрибуты: ранк отправителя и получателя (на разных концах соответственно), адрес в памяти от куда начинается сообщение и куда его следует поместить, тип данных в сообщение и его размер. При этом MPI гарантирует доставку сообщения, а программист должен гарантировать, что все отправленные сообщения будут получены.

Слайд 16
Самый простой тип взаимодействия — это взаимодействия типа точка-точка, когда один процесс передает сообщение, а другой его получает. При этом передача может быть как синхронной, так и асинхронной.

Слайд 17

При синхронной передаче процесс отправитель в передающей функции остается до тех пор, пока он не получит подтверждение, что сообщение получено.

Слайд 18
При асинхронной передаче отправитель уведомляется, что сообщение отправлено, но в дальнейшем необходимо проверить, что сообщение доставлено.

Слайд 19
Синхронной/блокируещей может быть не только отправка, но и прием сообщения. Время блокировки представлено на слайде.

Слайд 20
Асинхронным/неблокирующим также может быть как прием так и передача. Стоит обратить внимание, что при неблокирующих операциях некоторые ресурсы не должны использоваться до момента подтверждения завершенности операции.

Слайд 21
Помимо взаимодействия точка-точка в MPI определены коллективные операции, которые вовлекают все подпрограммы в группе. Конечно, эти операции могут бы реализованы через операции точка-точка, но во многих случаях они оптимизированы в плане передачи группового взаимодействия.

Слайд 22
Broadcast — рассылка одного сообщения всем подпрограммам. Так на слайде процесс P1 рассылает данные другим трем процессам.

Слайд 23, 24
Scatter — распределение данных между процессами, a gather — сбор порций данных в один единый массив.

Слайд 25
Если в случае gather над данными надо выполнить еще какую-то операцию (например посчитать сумму), то для этих целей в VPI предусмотрена операций глобальной редукции. Есть вариант, когда результат редукции рассылается всем процессам, т.е. операция редукции объединена с операцией broadcast.

Слайд 26
Последняя групповая операция — это барьер. Достигнувшие барьера процессы дожидаются пока к барьеру не подойдут все остальные и лишь затем все дружно продолжают свою работу.

Слайд 27
Стандарт определяет правило именования MPI типов и функций. Правило именования представлено на слайде. Все константы в MPI пишутся заглавными буквами. Регистр букв относится только к C/C++.

Слайд 28
После запуска MPI программы должна пройти инициализация всей группы процессов прежде чем появится возможность обмениваться сообщениями. Для этого следует вызвать функцию MPI_Init с указанными аргументами. Это должна быть первая MPI функция в программе.

Слайд 29
Последней MPI функцией должна быть MPI_Finalize. Каждая подпрограмма должна вызвать функцию инициализации и завершения. Только после того как все подпрограммы вызовут функцию завершения, MPI программа завершиться и узлы кластера освободятся.

Слайд 30
Подпрограммы MPI программы можно объединять в группы. Группа, в которую входят все подпрограммы называется MPI_COMM_WORLD. Группы и порядок именования подпрограмм в группе называется коммуникатором.

Слайд 31
Как уже упоминалось, подпрограмма в группе (коммуникаторе) идентифицируется ранком. Получить ранк можно с помощью функции MPI_Comm_rank().

Слайд 32
Размер группы можно узнать с помощью вызова MPI_Comm_size(). При этом с параметром MPI_COMM_WORLD можно узнать количество процессов, которое пользователь запросил при запуске MPI программы (ключ -np).

Слайд 33, 34
На слайде представлен код программы, которая печатает свой ранк и количество запрошенных пользователем процессов. Приведена строка компиляции и запуска программы. Видно, что от запуска к запуске вывод может отличаться.

Слайд 35
При передаче данных необходимо указывать тип данных, из которых состоит сообщение. Есть предопределенные типы данных, которые имеют прямое отображение на базовые типы данных языка программирования. Есть возможность из этих базовых типов данных конструировать свои типы.

Слайд 36, 37
Предопределенные типы данных для C/C++ и Фортран.

Слайд 38
Передача сообщения осуществляется с помощью функции MPI_Send(), прототип которой представлен на слайде. Стоит отметить, что помимо самого сообщения, его типа, размера и получателя вводится еще один параметр tag, который может использоваться для того, чтобы различать сообщения от одного отправителя.

Слайд 39
Получить сообщение можно с помощью функции MPI_Recv(). Информация о полученном сообщение возвращается в структуру MPI_Status.

Слайд 40
Для успешного завершения передачи сообщения должны быть корректно указаны ранки получателя и отправителя в одном и том же коммуникаторе, размер буфера для приема сообщения должен быть больше или равен размеру сообщения. Таги также должны совпадать.

Слайд 41
Пример передачи сообщения от подпрограммы с ранком 0 подпрограмме с ранком 1

Слайд 42
Иногда возникают ситуации, когда неизвестно от кого должно первым придти сообщение и с каким ранком. В таких случаях можно указывать принимать сообщение от любого отправителя и/или с любым ранком. Реальный таг и ранк можно узнать из структуры MPI_Status.

Слайд 43, 44
Помимо ранка источника и тага в структуре возвращается размер сообщения, который можно получит через вызов MPI_Get_count().

Слайд 45
Рассмотренные функции передачи и приема сообщения являются блокирующими. Рассмотрим их неблокирующие варианты.

Слайд 46
Используя уже рассмотренные функции при попытке реализовать представленную последовательность передачи сообщения мы введем программу в состояние мертвой блокировки. Программа не выйдет никогда из функции MPI_Send, поскольку сообщение на другом конце не будет получено.

Слайд 47
Чтобы избежать в таком случае мертвых блокировок можно использовать неблокирующую функцию отправки сообщения. Все неблокирующие функции имеют префикс MPI_I. Они инициализирую передачу/прием сообщений и связывают этот запрос с некоторым дескриптором, по которому в дальнейшем можно проверить статус завершения операции. Могут использоваться для совмещения операций передачи и вычислений.

Слайд 48, 49
Прототипы неблокирующих функций приема и передачи представлены на слайде. Функция MPI_Wait позволяет дождаться завершения операции.

Слайд 50
Проверка статуса завершения может также быть блокируемой (MPI_Wait) или неблокируемой (MPI_Test).

Слайд 51
Существуют варианты проверки завершенности сразу нескольких операций.

Слайд 52
Подводя некоторый итог можно сказать, что функции передачи и приема сообщений могут быть как блокирующие так и неблокирующие. При этом неблокирующий send может использоваться с блокирующим recv и наоборот.

Мы не рассматриваем такие вариации функций передачи как синхронная, буферизированнаяи ряд других передач.

Слайд 53
При приеме и передаче сообщений необходимо указывать тип данных. Тип данных может быть как базовый тип MPI, так и пользовательский, который сконструирован из базовых типов. Пользователь может конструировать вектора, массивы, структуры.

Слайд 54
Конструируя новый тип, пользователь указывает последовательность базовых типов и их смещение от начала нового типа.

Слайд 55
Пример структуры как пользовательского типа представлен на слайде. Смещение указано в байтах.

Слайд 56
Самым простым пользовательским типом является массив — непрерывная последовательность фиксированной длинны из базовых типов (или других пользовательских типов). Прототип функции для определения такого типа данных представлен на слайде.

Слайд 57
Отличие вектора от массива в том, что между реальным расположением данных может быть некоторый зазор. Прототип функции для для определения такого типа представлен на слайде.

Слайд 58
Немного сложнее выглядит прототип для определения структур.

Слайд 59
В качестве примера разберем создание простой структуры из двух переменных одна из которых int, а вторая char. В зависимости от компилятора размер структуры может быть большим или равным сумме размеров этих двух типов. Если при передаче требуется передавать массив структур, то программисту необходимо позаботится о выравнивании границ структуры на границу слова с тем чтобы размер нового MPI типа совпадал с размером типа C/C++.

Слайд 60
При формировании описания структуры смещение можно вычислять через взятие адреса, или через функцию MPI_Address().

Слайд 61
После того, как тип определен, его надо создать. Только после создания новый тип можно будет использовать.

Слайд 62, 63
Сконструированные вектора в памяти могут занимать больше места, чем реально в них полезной информации. Получить размер занимаемого пространства и реальных данных в типе можно через функции MPI_Type_size и MPI_Type_extent.

Слайд 64
Удобно, когда можно расположить (переименовать) процессы в соответствие с задачей, например, если решается уравнение на двумерной решетке, то хочется знать кто является соседом справа/слева/сверху/снизу. Еще большим плюсом будет являться адаптация расположения процессов к архитектуре кластера. Такое переименование может упростить код.

Слайд 65
Для этого необходимо с помощью MPI функций создать собственную топологию (определить новый коммуникатор).

Слайд 66
Выделяют два типа топологий: декартова (решетка) и граф.

Слайд 67
При создание декартовой топологи необходимо указать количество размерностей, размеры решетки по каждой их осей, следует ли соединять границы по каждой из координат. Так, например, можно создавать кольца, торы и т.д. Также можно попросить переупорядочить ранки подпрограмм.

Слайд 68
Для каждой программы в группе можно зная ее ранк получить ее координаты в решетке.

Слайд 69
Для определения соседа в декартовой топологии используется функция MPI_Cart_shift. При этом если соседа нет (граница), то возвращается MPI_PROC_NULL. На MPI_PROC_NULL можно выполнять посылку сообщения. При этом реальной передачи не будет.

Слайд 70
Создание топологии типа граф осуществляется с помощью функции MPI_Graph_create().

Слайд 71
В коллективных операциях участвуют все процессы в коммуникаторе. К коллективным относятся Broadcast, Barier, Scatter, Gather и глобальная редукция.

Слайд 72
При этом все процессы обязаны вызвать коллективную функцию. У коллективной функции нет тэга, и все коллективные функции являются блокирующими.

Слайд 73
Синхронизация подпрограмм может осуществляться с помощью функции MPI_Barier(), которая имеет всего один аргумент — коммуникатор. Эту функцию рекомендуют использовать только для отладки и профилирования программы. Синхронизация подпрограмм через посылку сообщения может быть более эффективной.

Слайд 74
MPI_Broadcast() - рассылка одного сообщения всем имеет большее количество аргументов. Так ранк того, кто рассылает сообщение указывается в качестве root-а. Для этой подпрограммы buf является входным параметром, а для остальных выходным.

Слайд 75
MPI_Scatter() - разрезание блока на части и рассылка частей по подпрограммам. Root — ранк подпрограммы, которая выполняет рассылку. Каждой подпрограмме отправляется по sendcount элементов типа sendtype. Таким образом i-ый участок sendbuf отправляется i-ой подпрограмме.

Слайд 76
MPI_Gather() - операция обратная MPI_Scatter.

Слайд 77
Глобальная редукция реализуется только с коммутативными операциями. Есть предопределенные операции и пользователю позволено создавать свои.

Слайд 78
Предопределенные операторы редукции представлены на слайде.

Слайд 79
Прототип функции глобальной редукции представлен на слайде. Опять же root — указывает кому должен возвращаться результат. Есть вариант функции, когда результат отсылается всем подпрограммам.

Слайд 80
В курсе были рассмотрены следующие вопросы: Модель процессов MPI, способы передачи информации как точка-точка: так и коллективные. Последние упрощают код и оптимизируют передачу сообщений между подпрограммами. Были рассмотрены возможности по созданию собственных типов данных и виртуальных топологий.

Курс охватывает лишь небольшую часть функций определенных в стандарте. Полный их перечень с примерами можно найти в документации.

Слайд 81, 82, 83
Шаблоны программ.