Стандарт параллельного программирования MPI

Введение

Рассмотренный на прошлых занятиях стандарт параллельного программирования OpenMP предназначен для компьютеров с так называемой разделяемой памятью. Это означает, что у такого компьютера имеется несколько процессоров (или один процессор с несколькими ядрами), имеющих равные возможности для доступа к любой части оперативной памяти компьютера. Такие компьютеры встречаются сейчас очень часто, однако их возможности в плане параллелизма ограничены. На практике часто встречаются двух-, реже четырех-, и совсем редко более чем четырехъядерные процессоры. Дело здесь не только в стоимости, но и в том, что с увеличением числа процессоров или числа ядер одного процессора, при сохранении принципа равных возможностей доступа к памяти, резко усложняется схема связи процессоров с модулями памяти, и все большая часть времени начинает расходоваться не на полезную работу, а на разрешение конфликтов, когда два процессора одновременно обращаются к одному модулю памяти. Примерно по этой же причине неразумно подключать к сети Ethernet слишком много компьютеров и при увеличении размеров локальной сети требуется ее структуризация, т. е. выделение небольших фрагментов и организация соединений между ними.

Для достижения гораздо более высокой степени параллелизма используется другой принцип построения компьютеров, который называется «компьютер с распределенной памятью» или «кластер». В этом случае мы имеем несколько (обычно десятки или даже сотни, реже тысячи) вычислительных модулей, каждый из которых представляет собой почти отдельный самостоятельный компьютер со своим процессором и памятью. Такие вычислительные модули объединяются при помощи высокоскоростной локальной сети и общаются друг с другом путем посылки так называемых сообщений по этой сети.

Компьютеры с распределенной памятью отличаются от компьютеров с разделяемой памятью тем, что время доступа какого-либо процессора к «своей» памяти, т. е. к памяти, физически расположенной в том же вычислительном модуле, гораздо меньше, чем к «чужой». Поэтому для эффективного использования кластеров необходимо делить задачу на части таким образом, чтобы минимизировать «общение» между разными вычислительными модулями, т. е. объем пересылаемых между ними сообщений.

Посылка сообщений используется в кластерах не только для обмена информацией, но и для синхронизации. Поскольку использование синхронизации приводит к простою некоторых вычислительных модулей, также желательно уменьшать как количество синхронизационных сообщений, так и вред простоя, происходящий от использования оставшихся (разумеется, следя за сохранением работоспособности программы!).

Стандарт MPI ориентирован на параллельное программирование для компьютеров с распределенной памятью. Модель вычислений, принятая при использовании данного стандарта, такова: одна и та же программа загружается на несколько вычислительных модулей и выполняется на них, образуя неколько процессов. Существует функция, позволяющая конкретной копии программы (конкретному процессу) узнать свой номер среди всех запущенных таким образом процессов, сделанных из данной программы, а также еще одна функция, позволяющая узнать общее число таких процессов (по существу, так же, как и в OpenMP при программировании на низком уровне). По этой информации программа должна решить, какую часть общей работы нужно выполнять конкретно этому процессу, т. е. копии программы с конкретным полученным номером, а также, как, и с какими другими процессами ему нужно общаться — какие данные посылать, какие принимать и в какой синхронизации участвовать.

В отличие от OpenMP, в MPI в принципе не бывает разделяемых переменных — все данные передаются от одного процесса другому через обмен сообщениями (в последних версиях стандарта, правда, появилось понятие разделяемой памяти).

Дальнейшая часть введения посвящена подготовке к работе с параллельными программами, использующими реализацию MPICH стандарта MPI, на компьютерах кластера в аудитории 2444 математико-механического факультета СПбГУ.

1. Имена компьютеров, перезагрузка в Linux и вход. Пользователям кластера доступны 18 компьютеров; их имена m01, m03, ..., m10 (класс у окна) и l01, l02, ..., l09 (класс у двери) написаны

на их корпусах (у двери написаны на корпусах s..., а имена l...). Компьютер m02 не существует; попытки его использования приводят к сообщениям об ошибках.

Обычно, на этих компьютерах работает операционная система Windows Server 2003. Для работы с MPICH компьютеры должны быть перезагружены в Linux. Это делается в два этапа: чтобы начать процесс перезагрузки, нужно нажать мышью кнопку «Shut down ...» в окне, приглашающем к входу в Windows, и в предлагаемом диалоге выбрать Restart (обычно этот вариант уже выбран), и нажать «Ok». После того, как Windows завершит свою работу и начнется повторная загрузка, пользователю будет предложено меню с выбором из двух пунктов: Windows и SUSE Linux. Данное меню позволяет выбрать желаемый вариант в течение нескольких секунд, и если пользователь за это время не выбрал Linux, автоматически снова загружается Windows.

Когда загрузится Linux, нужно войти в систему, используя те же имя и пароль, которые Вы используете для входа в Windows.

Для ввода указанных далее команд нужно сначала запустить эмулятор терминала. Это делается следующим образом: щелчок левой кнопкой мыши по зеленому круглому значку в левом нижнем углу экрана (где у Windows кнопка Start), выбор первой вкладки и на ней (обычно нижняя строка, нужно прокрутить содержимое этой вкладки вниз до конца) значок терминала и надпись «Консоль».

После работы желательно опять перезагрузить компьютер в Windows, для чего в консольном режиме достаточно нажать Ctrl-Alt-Del, а в оконном — щелкнуть левой кнопкой по зеленому круглому значку в левом нижнем углу экрана, в появившемся меню выбрать последнюю вкладку и в ней — пункт «перезагрузить».

2. Подготовка ssh. Во время работы MPICH при использовании компьютеров, отличных от того, на котором зарегистрировался пользователь (а параллельность предполагает такое использование), MPICH вынужден выполнять определенные команды на этих компьютерах от имени пользователя. Для этого используется средство ssh (шифрующий аналог rsh, средства для выполнения команд на удаленном компьютере). Поэтому для правильной работы MPICH ssh должен быть настроен так, чтобы при его использовании в пределах кластера пользователь мог входить на другие компьютеры без ввода пароля.

Для этого пользователь, зарегистрировавшись на одном из компьютеров кластера, должен сделать следующее:

а) Создать при помощи утилиты ssh-keygen пару из секретного и открытого ключей. Для этого нужно дать команду

```
$ ssh-keygen
```

Здесь и далее знак $ является частью приглашения к вводу команды, и набирать его не нужно. Утилита ssh-keygen во время своей работы предлагает пользователю ввести секретную фразу для защиты ключа; поскольку в данном случае такая фраза не нужна, в ответ на этот вопрос нужно просто нажать «ввод». Этот вопрос повторяется дважды.

Выполнение этой команды приведет к созданию в домашнем директории поддиректория .ssh, и в нем файлов id_rsa (секретный ключ) и id_rsa.pub (открытый ключ).

б) Далее, необходимо перейти в директорий .ssh (например, командой cd .ssh) и создать в нем файл authorized_keys, позволяющий входить на другие компьютеры кластера без пароля. Это делается в два этапа. Во-первых, необходимо создать файл, в котором 18 раз повторяется строка из файла id_rsa.pub (последний файл состоит из одной строки). Это можно сделать, например, такой командой:

```
$ for((I=0; I<18; I++)); do cat id_rsa.pub >> authorized_keys; done
```

Далее, полученный файл authorized_keys необходимо отредактировать в любом текстовом редакторе, заменив в конце каждой строки в адресе логин@имякомпьютера имя компьютера так, чтобы во всех 18 строках встречались имена всех 18 компьютеров (в остальном содержимое этого файла должно остаться без изменений). Для этого см. раздел 3 — редактирование программы.

в) Наконец, нужно при помощи ssh (командой ssh имякомпьютера) зайти на каждый из компьютеров кластера (в том числе и на тот, на котором изначально зарегистрирован пользователь). Это

делается по двум причинам: во-первых, при первой такой регистрации ssh спрашивает, верен ли открытый ключ того компьютера, на котором мы собираемся регистрироваться. На этот вопрос надо ответить yes, и при следующих попытках регистрации этот вопрос уже не задается. Во-вторых, это позволяет проверить правильность действий, выполненных на предыдущих этапах — ssh не должен спрашивать пароль при такой регистрации.

Если все было сделано правильно, и зарегистрированному на одном компьютере кластера пользователю ssh позволяет регистрироваться на любом компьютере кластера без пароля, можно переходить к следующему пункту.

3. Редактирование программы. В Linux имеется достаточно много текстовых редакторов, позволяющих ввести программу и исправить имеющиеся в ней ошибки. Мы рассмотрим один из простейших вариантов, а именно, встроенный редактор оболочки mc. Оболочка mc очень похожа на FAR из Windows. Для ее запуска нужно дать команду

```
$ mc
```

После запуска экран делится вертикально на две панели, в одной из которых отображается список поддиректориев и файлов текущего директория. Для редактирования уже имеющегося файла нужно поставить на него выделенную строчку (клавишами вверх/вниз) и нажать F4. Для создания нового (пустого) файла можно воспользоваться командой

```
$ > имя-файла
```

В редакторе можно набирать текст обычным образом, работают также клавиши Backspace, Del, стрелки, PageUp, PageDown. Выделить блок можно, поставив курсор на начало и нажав F3, затем поставив курсор на символ, следующий за последним символом, который должен быть в блоке, и нажав F3 еще раз. Блок можно записать на диск в отдельный файл (ctrl-F) или удалить (F8); поставив курсор на нужное место, можно скопировать (F5) или переместить (F6) блок туда. Можно вставить содержимое другого файла в позицию курсора (F15 или shift-F5).

Можно искать текст (F7) или искать и заменять (F4).

Команда undo — ctrl-u, запись файла — F2, выход — F10.

Эти и некоторые другие команды также можно вызвать из меню (F9).

Эмулятор терминала может перехватывать некоторые функциональные клавиши, понимая их как команды себе. Если это происходит, можно вместо F1, ..., F9, F10 нажимать (последовательно, а не одновременно) Esc и далее 1, ..., 9, 0. Т. е. F3 — то же, что и Esc 3.

4. Компиляция MPI-программы. Для компиляции MPI-программы, написанной на C, проще всего воспользоваться специальной версией компилятора, которая называется mpicc.

Например, скомпилировать программу prog.c из текущего директория можно командой

```
$ mpicc prog.c
```

При этом, если программа содержит ошибки (выявляемые при компиляции), mpicc сообщит пользователю о них. Если же ошибок нет, в текущем директории появится результат компиляции, исполняемая программа a.out (есть и компилятор C++, он называется mpiCC).

5. Запуск программы и замер времени. Если программа была успешно скомпилирована, ее можно запустить на выполнение командой

```
$ mpirun -np число -machinefile файл a.out
```

Здесь число обозначает требуемое число процессов, а файл — файл, содержащий список имен компьютеров, на которых допустимо выполнять эти процессы. Обычно, этот файл содержит имена всех доступных компьютеров из числа имеющихся в классе, по одному имени на строке. В этом файле, однако, должны быть имена только тех компьютеров, которые включены и на которых загружена операционная система Linux. Иначе система mpich может выдавать сообщения об ошибках.

Для замера времени работы программы можно воспользоваться средством time из командного интерпретатора:

```
$ time mpirun -np число -machinefile файл a.out
```

Команда time в конце работы программы выводит результаты трех замеров времени: общее время (от начала работы программы до ее завершения), время работы собственно программы на том компьютере, на котором она была запущена, и время работы операционной системы по обслуживанию пользовательской программы, также на том компьютере, на котором работает пользователь. Время выводится в минутах и секундах с точностью до 0,001 с. Нужно также понимать, что если программа содержит ввод с клавиатуры, время ввода также учитывается, поэтому для замера времени работы программы лучше писать такие варианты программы, которые ничего с клавиатуры не вводят.