Демьянович Юрий Казимирович

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ УРОК 2. О СРЕДСТВАХ РАСПАРАЛЛЕЛИВАНИЯ... ЭЛЕМЕНТАРНАЯ ПАРАЛЛЕЛЬНАЯ ПРОГРАММА

На прошлом уроке (см. [1]) мы узнали, что для решения суперсложных задач (для предсказания землетрясений, цунами и т. п.) требуется огромное количество арифметических действий, причем решать такие задачи нужно достаточно быстро, с тем чтобы успеть принять соответствующие меры (эвакуировать людей, включить аппаратную защиту и т. д.). Мы узнали также, что для решения таких задач нужны суперкомпьютеры, то есть такие вычислительные системы, которые обладают быстродействием, на много порядков превосходящим быстродействие обычных компьютеров. На современном уровне науки и технологических исследований лишь параллельные системы (вычислительные системы с большим числом параллельно действущих вычислительных модулей, процессоров или ядер) позволяют достичь такого быстродействия.

Однако остается неясным вопрос о том, как же написать параллельную программу, то есть программу, используя которую суперкомпьютер будет проводить вычисления параллельно. Вспомним, что при распараллеливании разные вычислительные модули суперкомпьютера производят различные вычисления, так что, вообще говоря, разным вычислительным модулям приходится работать по различным программам или использовать различные исходные данные. Вручную писать отдельные программы или выписывать различные данные для различных вычислительных модулей нецелесообразно, ибо количество таких модулей может достигать ста тысяч. Требуются специальные программные средства, которые обрабатывают поступающую программу и автоматически распределяют вычисления по различным вычислительным модулям.

Для распараллеливания последовательной программы на первом этапе нужно в ней выделить части, которые можно выполнять параллельно (то есть такие части, которые можно выполнять одновременно и независимо: между этими частями не должно быть взаимодействия в процессе их выполнения, а исходные данные для этих частей должны быть получены при выполнении предыдущих частей программы).

На втором этапе в программу в нужных местах вставляются команды (операторы или директивы) применяемого средства распараллеливания. Третий этап состоит в запуске и отладке полученной таким образом программы на параллельной вычислительной системе.

Для распараллеливания...

Наиболее сложным является первый этап – этап выделения частей исходной последовательной программы, которые можно обрабатывать независимо; именно эти части в дальнейшем (на втором этапе) оформляются с помощью упомянутых средств распараллеливания для превращения последовательной программы в параллельную. При этом программист должен указать места в программе, которые, по его мнению, допускают эффективное распараллеливание, и описать общие приемы такого распараллеливания. Именно в этих местах упомянутые специальные программные средства возьмутся за распределение вычислений по параллельным вычислительным модулям в согласии с указанными приемами распараллеливания.

Возникает вопрос: нельзя ли поручить упомянутым программным средствам более глубокий анализ программы, с тем чтобы пользователю-программисту не приходилось указывать места в программе, пригодные для распараллеливания? Окончательного ответа на такой вопрос нет, однако заранее можно сказать, что такие программные средства должны будут обладать большой интеллектуальной мощью; в настоящее время подобные средства отсутствуют. Это утверждение иллюстрируется примерами, приведенными в книге [2: с. 385–390] и в курсе лекций [3: с. 126–131]; речь идет о распараллеливании двойного цикла на двухпроцессорной вычислительной системе. Оказывается, что, несмотря на кажущуюся простоту этого примера, ответить на вопрос о возможности распараллеливания (с сохранением того вычислительного результата, который получается на однопроцессорной системе) достаточно сложно: требуется глубокий анализ ситуации и учет дополнительных свойств вычислительной системы.

Возникает вопрос: что следует потребовать от специальных программных средств распараллеливания?

Желательно, чтобы эти средства обладали следующими свойствами:

1) простотой использования (небольшим количеством достаточно эффективных команд-директив),

2) применимостью к уже имеющимся последовательным программам, с тем чтобы эти программы можно было превратить в параллельные и запустить на параллельной системе,

3) согласованностью с языками высокого уровня (поскольку именно такие языки используются пользователями-программистами, часто являющимися специалистами в своей предметной области, но отнюдь не системными программистами).

Кроме того,

4) важно, чтобы для использования средств распараллеливания программисту не требовалась информация о параллельной вычислительной системе (о характеристиках и о количестве ее параллельных вычислительных модулей, об используемой разрядной сетке и т. п.),

5) с другой стороны, желательно, чтобы были средства постепенного совершенствования параллельной программы по мере накопления информации (а именно, информации об «узких» местах программы, ведущих к замедлению ее работы, о характеристиках используемой вычислительной системы и т. д.),

6) желательна наглядность получаемой программы, с тем чтобы было легко отслеживать ее смысл и структуру (наиболее часто программисты страдают от того, что через один-два месяца забывают смысл программы: приходится тратить много времени на изучение своей собственной программы; заметим, что, как правило, дополнительное описание программы и комментарии в ее тексте не спасают от этой неприятности),

7) желательно, чтобы способ обращения к средствам распараллеливания был стандартизован и применялся одинаковым образом на различных параллельных системах, для того чтобы написанную программу (иногда весьма сложную) не требовалось переписывать заново при переходе на параллельную вычислительную систему другого типа (то есть чтобы поддерживалась переносимость программ),

8) поскольку процесс распараллеливания может пошагово совершенствоваться различным образом (при этом могут быть затронуты принципы, положенные в основу исходной последовательной программы), неплохо было бы иметь возможность вернуть-

ся к предыдущему шагу, в том числе и к исходной (или модифицированной) последовательной программе.

Конечно, эти пожелания минимальны; кроме них хочется иметь эффективные средства отладки программ, использующих рассматриваемые программные средства, наглядные средства наблюдения за параллельными потоками и их обработкой, простые средства корректировки параллельных потоков (то есть фактически, наглядные средства модификации процесса распараллеливания, которые генерируют текст параллельной программы) и многое другое.

Что же сделано к настоящему моменту?

При ответе на этот вопрос заметим, что развитие специальных программных средств распараллеливания происходит во всех перечисленных направлениях, но ни одно из них не находится на завершающем этапе. Напомним, что речь идет о суперкомпьютерных вычислениях, что постоянно появляются новые суперкомпьютеры (список TOP500 самых мощных суперкомпьютеров мира обновляется два раза в год), а практически для каждой новой параллельной системы нужно создавать свое собственное программное обеспечение (в том числе, писать транслятор с языка высокого уровня и специальные программные средства распараллеливания со стандартным обращением к ним для поддержания переносимости уже созданных параллельных программ). Заметим, что именно по этой причине специальные средства распараллеливания разрабатываются обычно для алгоритмических языков *C*, *C++* и *Fortran* (эти языки наиболее часто применяются для решения суперсложных задач).

Одним из наиболее распространенных стандартов параллельного программирования является стандарт MPI; согласно принятому в мире соглашению, каждая параллельная система должна иметь в своем программном обеспечении некоторую реализацию этого стандарта. Аббревиатура MPI расшифровывается как Message Passing Interface (то есть интерфейс передачи сообщений: слово интерфейс означает, что речь идет о стандарте обращения к рассматриваемому программному средству и о его требуемых свойствах, но не о стандарте его реализации – естественно, что способ его реализации для каждой параллельной системы не может быть стандартным). Это специальное программное средство распараллеливания было создано в Исследовательском центре фирмы IBM в 1994 году, а в 1997 году появилась вторая его версия; стандарт MPI можно рассматривать как библиотеку программ распараллеливания, добавляемую к алгоритмическим языкам *Fortran*, *C* или *C++*.

Программа на алгоритмическом языке *Fortran* с использованием MPI имеет обычный вид, где об использовании MPI говорят только присутствующие в программе вызовы процедур стандарта MPI, оформленных как процедуры библиотеки `mpif.h` (такую программу обычно называют MPI-программой).

Приведем простой пример такой программы (ее строки прокомментированы ниже).

Основными понятиями MPI являются:

– процесс;

– группа процессов;

– коммуникатор.

Коммуникатор объединяет (и идентифицирует) группу процессов, которые с точки зрения данного коммуникатора рассматриваются как параллельно исполняемые последовательные программы; последние, однако, могут на самом деле представлять группы процессов, вложенных в исходную группу.

MPI допускает многократное ветвление программы и создание на базе одного про-

Коммуникатор объединяет...

цесса очередной группы процессов, объединяемых новым коммуникатором. Таким образом, процесс ветвления может представлять собой дерево; для завершения программы не требуется возвращение к исходному процессу.

Коммуникатор производит синхронизацию и обмены между идентифицируемыми им процессами; для прикладной программы онвыступает как коммуникационная среда. Каждый коммуникатор имеет собственное коммуникационное пространство, а сообщения,использующие разные коммуникаторы, не оказывают влияния друг на друга и не взаимодействуют. Таким образом, каждая группа процессов использует свой собственный коммуникатор, процессы внутри группы нумеруются от 0 до $k-1$, где k – число процессов в группе (параметр k может принимать различные натуральные значения, задаваемые пользователем, но не превосходящие значения, определяемого реализацией).

MPI управляет системной памятью для буферизации сообщений и хранения внутренних представлений объектов (групп, коммуникаторов, типов данных и т. п.)

В структуре языков *Fortran*, *C* или *C++* стандарт MPI представляется как библиотека процедур с вызовами определенного вида. Требуется, чтобы программа, написанная с использованием стандарта MPI, могла быть выполнена на любой параллельной системе без специальной настройки.

Для применения MPI на параллельной системе к программе должна быть подключена библиотека процедур MPI (в приведенной выше программе – это библиотека mpif.h, см. листинг 1).

Перед использованием процедур стандарта MPI следует вызвать процедуру MPI_INIT; она подключает коммуникатор MPI_COMM_WORLD, задающий стандартную коммуникационную среду с номерами процессов 0, 1, ..., $n-1$, где n – число процессов, определяемых при инициализации системы. При завершении использования MPI в программе должна быть вызвана процедура MPI_FINALIZE; вызов этой процедуры обязателен для правильной работы программы.

Следует представлять себе ситуацию таким образом, что рассматриваемая программа скопирована во все вычислительные модули параллельной системы. Прокомментируем программу, представленную выше (см. листинг 1):

– строка 1 не нуждается в объяснении;

– строка 2 подключает библиотеку процедур MPI под именем mpif.h, в которой, в частности, содержатся процедуры MPI_INIT, MPI_COMM_WORLD, MPI_FINALIZE;

– строка 3 вызывает MPI_INIT для инициализации MPI;

– строка 4 возвращает число процессов nprocs, принадлежащих коммуникатору MPI_COMM_WORLD;

– строка 5 возвращает ранг (номер) myrank процесса внутри стандартной коммуникационной среды (в данном случае абстрактно его можно представлять как номер вычислительного модуля, на котором происходит данный процесс);

– строка 6 производит распечатку вычисленных параметров nprocs и myrank для данного процесса (в данном случае можно представлять себе дело таким образом, что каждый вычислительный модуль параллель-

Листинг 1. Простой пример программы на языке Fortran, использующей стандарт MPI

```
envelope.f
1)   PROGRAMM envelope
2)   INCLUDE 'mpif.h'
3)   CALL MPI_INIT(ierr)
4)   CALL MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
5)   CALL MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
6)   PRINT *,'nprocs =',nprocs,'myrank =',myrank
7)   CALL MPI_FINALIZE(ierr)
8)   END
```

ной системы имеет свой принтер, на котором и производится распечатка); в результате каждый вычислительный модуль сообщит номер процесса, которым он занят;

– строка 7 вызывает процедуру MPI_FINALIZE, завершающую работу с библиотекой MPI (вызов ее обязателен и после ее вызова взаимодействие с MPI невозможно).

Приведенные комментарии дают определенное представление о работе процедур MPI_INIT, MPI_COMM_WORLD, MPI_FINALIZE, а также процедур MPI_COMM_SIZE, MPI_COMM_RANK; более подробные описания процедур интерфейса MPI и примеры их использования будут даны в следующих лекциях.

Заметим, что при запуске программы пользователю не нужно ничего знать о свойствах параллельной системы: ни о числе параллельных вычислительных модулей, ни об их индивидуальных свойствах, ни об общей архитектуре системы и т. п. (имеется возможность запускать MPI-программы на однопроцессорной системе, имитируя распараллеливание; такая возможность обычно используется для отладки MPI-программы перед запуском ее на параллельной системе).

В заключение предлагается ответить на следующие вопросы.

1. Почему все суперкомпьютеры являются параллельными системами?

2. Для чего следует заниматься параллельными вычислениями?

3. Из-за чего требуются специальные программные средства при распараллеливании последовательной программы?

4. Назовите этапы распараллеливания последовательной программы.

5. Какой из этапов распараллеливания является наиболее сложным?

6. Почему не удается полностью автоматизировать процесс распараллеливания последовательной программы?

7. Какими качествами должны обладать программные средства распараллеливания? Какое из этих качеств является наиболее важным?

8. Когда появился стандарт MPI? Какая организация разработала этот стандарт?

9. Зачем нужно было принятие единого стандарта распараллеливания?

10. Каковы основные понятия стандарта MPI?

11. Как вы представляете реализацию распределения работ между отдельными процессами при использовании стандарта MPI?

12. Можно ли представить ветвление программы (при использовании MPI) в виде дерева?

13. Имеется ли априори выделенный процесс в MPI-программе?

14. Требуется ли слияние всех процессов при завершении MPI-программы?

15. Нужна ли специальная настройка MPI-программы при ее перенесении с одной параллельной системы на другую?

16. Как вы думаете, что произойдет, если в MPI-программе будет отсутствовать вызов процедуры MPI_FINALIZE?

17. В чем состоит работа процедур MPI_COMM_SIZE и MPI_COMM_RANK?

Литература

1. *Демьянович Ю.К.* Параллельные вычисления. Урок 1. Параллельная форма // Компьютерные инструменты в школе, 2011. № 1. С. 36–39.

2. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб: БХВ-Петербург, 2002. 608 с.

3. *Бурова И.Г., Демьянович Ю.К.* Алгоритмы параллельных вычислений и программирование. Курс лекций. СПб: Изд-во С.-Петерб. ун-та, 2007. 200 с.

Демьянович Юрий Казимирович, доктор физико-математических наук, профессор, заведующий кафедрой параллельных алгоритмов математико-механического факультета СПбГУ.