О прикладных задачах и кластерных системах

Компьютерный мир меняется. Пятидесятилетняя эпоха последовательной организации вычислений сменяется эрой параллелизма, параллельных вычислительных технологий и параллельных вычислительных систем. Сначала появление кластерных систем, а затем переход всех ведущих производителей вычислительной техники на многоядерные процессоры, сделали компьютерный мир параллельным.

Революционность происходящего в наше время сродни изменениям, которые потребовались обществу в середине прошлого века для адекватного восприятия первых компьютеров. Сегодня задача не менее сложная, поскольку необходимо адаптироваться к параллельным компьютерным технологиям. Нужно пересматривать подходы к исследованию свойств и анализу качества алгоритмов, необходимо менять мышление программистов, нужен целый спектр специального инструментария для разработки программного обеспечения нового поколения.

Доступность современных параллельных компьютерных систем создает иллюзию легкости их использования. Многопроцессорные серверы, кластеры, многоядерные процессоры, распределенные вычислительные среды действительно можно использовать для решения любых задач, однако доступно – не значит просто. В параллельных вычислительных системах на передний план выходит понятие, на которое не так сильно обращали внимание в период господства последовательных методов организации вычислительного дела: эффективность использования компьютеров. В самом деле, а почему бы не сохранить традиционные последовательные технологии программирования? Берем существующую и уже работающую программу, из всего множества доступных процессоров, ядер или вычислительных узлов параллельного компьютера задействуем

только один экземпляр, компилируем, запускаем и получаем результат. Ничего сложного, все отработано годами, все привычно и никаких дополнительных проблем или неудобств эти шаги не вызывают. В принципе, все это верно, за исключением одного – забыто главное. Зачем создаются параллельные компьютеры? Для увеличения их производительности. Именно это дает возможность уменьшить время работы программ, перейти к новым размерам и размерностям в постановке задач, повысить точность их решения. И все это останется в стороне, если забыть про параллелизм и по старинке ориентироваться только на последовательные методы работы. Выполнить программу на одном процессоре, безусловно, можно, и человек даже сможет сказать, что он “работал на многопроцессорной системе”. Только какой в этом смысл? Нужно ли было тратить значительные средства на установку дорогостоящей системы, если эффект от нее такой же, как от обычного компьютера? Вряд ли. Целесообразно ли использовать параллельную вычислительную систему в таком режиме? Скорее всего – нет. Инструмент определяет методы и технологии его использования. Если в архитектуре компьютера присутствует параллелизм, значит он в том или ином виде должен быть и в программе. В противном случае, ответ на вопрос: “Зачем использовали параллельный компьютер?”, нужно искать в какой-то совсем иной плоскости.

Все сказанное в полной мере относится и к кластерным системам. Появившись в середине 90-х годов прошлого века как доступная альтернатива традиционным суперкомпьютерам, они показали себя достойным средством решения больших задач. Необычайно быстрый рост популярности кластеров в вычислительном сообществе объясняется двумя причинами: очень высоким показателем отношения пиковой производительности к стоимости и большой свободой при выборе конфигурации кластерной системы. Сегодня за весьма умеренную цену организация может установить систему значительной производительности, причем в той конфигурации, которая лучше всего подходит для решения ее

задач. Подобная ситуация сама по себе не может не радовать, однако нельзя забывать, что кластер – это инструмент, это одно звено в длинной цепочке решения задач на параллельных вычислительных системах. Звено, безусловно, важное, но не единственное. Создавая методы решения задач на параллельных компьютерах, мы обязаны рассматривать всю цепочку в целом: от постановки задачи, описания алгоритма, выбора технологий параллельного программирования, до вопросов организации программно-аппаратной среды самой вычислительной системы и ее инженерной инфраструктуры. Слабость любого одного звена может привести к резкому падению эффективности решения задачи в целом, и только согласованность решений, принятых на всех этапах, даст надежду на действительно хороший результат.

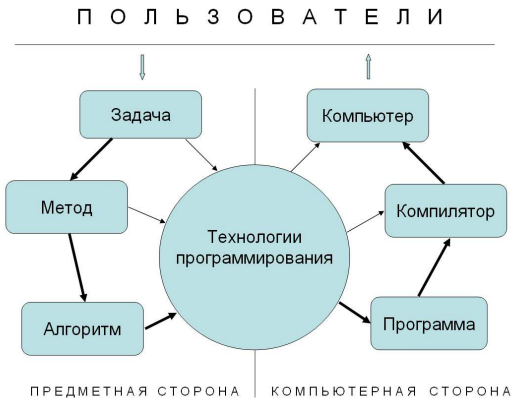

Рис. 1. Этапы решения задач на компьютерных системах

Обратимся к схеме решения задач (рис. 1) с помощью различного рода компьютерных систем. Стартовой точкой является задача – это то, что мы хотим найти, узнать или вычислить с помощью компьютера, это наша цель. Для решения задачи выбирается некоторый метод, т.е. математически обоснованная последовательность действий, следуя которой мы решение

получим. Под действием может пониматься как элементарная арифметическая операция, так и что-то более сложное, но не столь важное с точки зрения самого метода, например, вычисление скалярного произведения или нахождение максимального значения в заданном наборе чисел. Зафиксировав метод, следующим шагом необходимо описать алгоритм. Для этого нужно указать точный набор и порядок выполнения всех операций. Обычно, на данном этапе не очень задерживаются, так как выбранный метод автоматически определяет идею алгоритма. Но только идею. На практике, два алгоритма, построенные по одному и тому же методу, могут обладать принципиально разными свойствами, в частности, один может оказаться чисто последовательным, а другой алгоритм будет обладать значительным ресурсом параллелизма [1]. Двигаясь по цепочке далее, алгоритм записывается в виде программы с помощью той или иной технологии программирования, затем полученная программа обрабатывается компилятором, который уже и генерирует код, пригодный для исполнения компьютером. “Задача”, “Метод” и “Алгоритм” в большей степени относятся к предметной области, “Программа”, “Компилятор” и “Компьютер” составляют основу последующей реализации, а “Технологии программирования” – это посредник между двумя сторонами единого процесса решения задачи.

Такова стандартная цепочка, и на рисунке она отмечена жирными стрелками. Ее много раз проходил каждый программист, как системный, так и прикладной, выбирая оптимальный путь решения своей задачи. За счет использования специальных технологий программирования цепочка в ряде случаев может стать короче, позволяя пользователю найти решение его задачи проще и быстрее. В самом деле, воспользовавшись уже готовым прикладным программным комплексом, все решение можно свести лишь к правильному заданию входных данных, и путь от задачи через такие технологии программирования сразу приведет к компьютеру. Подобных “альтернативных” путей существует немало, и на рисунке они показаны тонкими стрелками.

Данная схема дает богатую пищу для размышлений и помогает понять причины многих проблем, возникающих при решении задач на компьютерах, особенно на компьютерах с параллельной архитектурой. Тема достойна отдельного обсуждения, однако сейчас сосредоточимся на этапе, который отвечает предмету данной книги – на кластерных системах. Это последнее звено цепочки, обозначенное на рисунке словом “Компьютер”, отвечающее за выполнение программ на кластере. Можно указать по меньшей мере три причины, почему данное звено занимает в цепочке особое положение и заслуживает особого внимания:

- тот факт, что программа будет работать на кластере, должен учитываться на всех предыдущих стадиях: это скажется на выборе метода и алгоритма решения задачи, технологий и систем программирования, т.е. на всех звеньях цепи;
- по мере прохождения этапов накопятся и именно здесь скажутся недостатки всех принятых ранее решений: не лучшие свойства метода, множество последовательных частей в алгоритме, плохо написанная программа, огрехи компилятора, и в результате – низкая эффективность работы параллельной программы;
- наконец, если не обеспечить качественного функционирования кластерной системы, лежащей в основе последнего звена цепочки, то станет просто бессмысленной вся работа, которая была выполнена на всех предыдущих этапах.

Приступая к проектированию кластерной системы, очень важно продумать всю цепочку в целом, чтобы сформировать будущее решение на основе целенаправленного выбора и анализа, а не ощущений или эмоций от рекламных кампаний. Общепринятые стандартные подходы хороши “в среднем”, но для конкретных задач они могут оказаться совершенно непригодными. Если заказчик просит спроектировать кластерную систему, но ничего не говорит о предполагаемом классе задач или же режиме использования системы, то это должно серьезно насторожить. В зависимости от того, создается ли центр коллективного пользования с

большим числом разных задач, учебный центр по параллельным вычислениям или вычислитель для поддержки конкретного проекта, в зависимости от того, насколько критичным является требование надежности и отказоустойчивости работы оборудования, сколь тесно кластер должен быть в будущем интегрирован в информационно-технологическую инфраструктуру предприятия – при каждом сценарии развития кластерного проекта нужно делать свой акцент в архитектуре комплекса. В противном случае, пострадает качество решения задачи, ради которой комплекс и создавался.

Подтверждений этому тезису в нашей практике очень много, и чем раньше удавалось уточнить требования и определить структуру задач, тем лучше получался результат. В одном проекте заказчик сразу же начал с того, что попросил проанализировать эффективность выполнения его программы на различных процессорах. Желание вполне обоснованное, если учесть, что кластерная система проектировалась только под одну конкретную задачу, и об универсальности установки речь не шла. Модельная программа была предоставлена, и нам оставалось лишь исследовать доступные серверные платформы, на основе чего и выбрать вариант построения будущего кластера. Был найден хороший вариант связки платформы и компилятора, дающей наименьшее время выполнения программы на различных наборах входных данных. Вроде бы задача выполнена, и можно было приступать к проектированию кластера в целом, однако настораживало немного необычное поведение модельной программы. Во время испытаний на каждой платформе программа занимала всю доступную оперативную память. Поговорили с заказчиком, и оказалось, что объем оперативной памяти во многом определяет и размер задачи, и время ее решения. Заказчик держал в голове “средние” параметры существующих кластерных систем, и минимизировал время работы программы за счет выбора подходящего процессора. По сравнению с предварительным проектом в окончательном варианте конфигурации кластера число вычислительных узлов было уменьшено в два раза, но зато

объем оперативной памяти на каждом узле был увеличен в четыре раза, что привело к значительному увеличению скорости решения исходной задачи. К слову, в окончательной конфигурации пришлось отказаться и от первоначально найденной “самой лучшей” платформы, которая не поддерживала эффективной работы с большими объемами оперативной памяти.

В другом проекте в научной группе планировался перевод большого программного комплекса, традиционно работавшего в однопроцессорном режиме, на параллельную кластерную систему. Нужно было переходить к новым размерам задач, но поскольку в существующем варианте время расчета было чрезмерно большим, то только переход на параллельную систему в перспективе мог дать необходимое ускорение работы программы. Составили проект, показали нам. Серьезных ошибок технического характера в проекте не было, а сказать что-либо еще без знания структуры будущих задач невозможно. Стали разбираться с программным комплексом и обнаружили потенциально серьезную проблему: интенсивная работа с файлами. При работе комплекса на одном процессоре этот вопрос и не возникал, так как обмен с дисками занимал около 1% всего времени работы программы. Однако переход на кластер из 32 процессоров с перспективой увеличения их числа в будущем до 128 заставлял серьезно задуматься о параметрах и конфигурации подсистем ввода/вывода, поскольку файловые операции все больше и больше выходили на первый план. Кстати, одно из определений суперкомпьютера в точности описывает данный эпизод: “Суперкомпьютер – это вычислительная система, сводящая проблему вычислений к проблеме ввода/вывода”. Немного иронично, но тонко подмечено. Когда вычислительная часть перестает быть узким местом, работу программы начинает сдерживать что-то еще, в данном случае, это работа с устройствами ввода/вывода. Очень жизненная ситуация, характерная для многих проектов.

Теперь рассмотрим типичный пример использования кластерных

систем, когда вся содержательная работа ведется через специализированные прикладные пакеты: ANSYS, GAMESS, ABAQUS, STAR-CD и другие. Нет сомнений, подход правильный и обоснованный: если кто-то уже подумал и о решении задач предметной области, и о распараллеливании, то готовым инструментом нужно воспользоваться. Нужно лишь быть уверенным в качестве двух принципиально важных компонентов. Первый – это сам пакет. Безусловно, он прекрасно справляется с определенными классами задач. Но все реальные задачи разные, пакеты же, как правило, ориентированы на некоторый “усредненный” вариант, и требуется специальная проверка их работоспособности при критических значениях параметров: предельных нагрузках, максимальных скоростях, переходных процессах, приграничных областях. Есть ли уверенность, что планируемый к установке на кластере пакет даст решение задачи? Вопрос непростой, нужна аккуратная оценка математики, методов, точности, диапазона входных данных и допустимых параметров моделей, заложенных в пакете. Второй компонент – это эффективность работы связки пакет-кластер на типичных наборах входных данных. Хорошо ли пакет масштабируется по числу процессоров? Может ли пакет использовать полностью имеющуюся оперативную память? Какие параметры коммуникационной сети для него наиболее критичны? Известно много случаев, например [2], когда конфигурация кластеров менялась после предварительных испытаний различных вычислительных платформ на тестах пакетов с учетом специфики входных данных, характерных для задач пользователей. Эффективно ли пакет адаптирован под архитектуру кластера? Данный вопрос также не следует оставлять без внимания, поскольку разработчики пакета могли использовать не совсем стандартные схемы и модели для реализации вычислительного процесса. Например, известный пакет для квантово-химических расчетов Gaussian для выполнения всех операций взаимодействия параллельных процессов использует систему Linda, которая сама по себе предполагает их общение через общую память. Необходимо убедиться заранее, существуют ли

эффективные реализации данного пакета на системах, аналогичных проектируемому кластеру. Иногда пакеты строятся на основе модели вычислительного процесса типа Мастер/Рабочие. В этом случае вся расчетная часть ложится на процессы-рабочие, а распределение работы между ними и управление процессом вычислений в целом выполняется процессом-мастером. Запуская пакет на четырех процессорах, нужно учитывать, что реальный счет будет идти только на трех, что определит и результирующее ускорение выполнения программы. Более того, в некоторых прикладных пакетах на каждый расчетный процесс порождается по одному вспомогательному, поэтому эффективно использовать 32 процессора реально сможет только программа, состоящая из 64 параллельных процессов.

Подобных нюансов на практике очень много. Завершая вводную часть книги, хочется еще раз подчеркнуть следующее. Главное в вычислительном деле – это задачи, в конечном итоге именно они определяют все. И чтобы перенести усилия пользователей в сторону предметной, содержательной и наиболее важной части их деятельности, нужно быть абсолютно уверенным в качестве используемых ими инструментов, в эффективности и согласованности всех этапов, через которые пользователи должны будут пройти на пути от постановки задачи до получения ее решения. Все этапы важны, ни один нельзя оставлять без внимания, каждый должен быть аккуратно исследован. Данная книга – о базовом уровне вычислительной практики, о кластерных системах, о тонких местах их проектирования, использования, сопровождения, о важных деталях и одновременно обо всех тех незаметных мелочах, которые скрасят вычислительные будни положительными эмоциями от красиво реализованных решений.