Кластерное программирование: не всегда можно получить желаемое

Опубликовано: 19 сентября 2006г., 4:27
Автор: Дуглас Идлайн
Но это не остановит меня задавать вопросы

Пятнадцать лет назад я написал короткую статью в ныне несуществующем журнале параллельных вычислений (Параллелограмм) под названием "Как вы Запрограммируйте 1000 процессоров?" В то время это был хороший вопрос, на который было нелегко ответить. Сегодня это еще хороший вопрос, на который до сих пор нет простого ответа. Кроме Сейчас это кажется немного более актуально, когда мы вступим в "Многоядерную" эпоху. В самом деле, когда я первоначально написал статью, до использования 1000 процессоров было далеко, но возможно. Сегодня, использование 1000 процессоров стало реальностью для многих практикующих HPC. Двух ядерный процессор попал в серверные комнаты, фактически удвоив скорость работы, гораздо больше людей будут присоединяться к 1000P клуб очень скоро.

Итак, давайте спросим: "Как вы будете программировать 10000 процессоров?" Как я понял, пятнадцать лет назад, такой вопрос не может действительно иметь полный ответ. В истории компьютеров, никто не ответил на вопрос по –настоящему - даже при рассмотрении десяти процессоров. Конечно, есть много методов и идей, как потоки, сообщения, барьерная синхронизация и т.д., но когда мне нужно больше думать о компьютере, чем о своей проблеме, что-то не так.

Потратив много ночей я пытался программировать параллельные компьютеры (последним воплощением были кластеры) я придумал список качеств, которые я хочу в язык параллельного программирования. Некоторые из особенностей я опишу ниже, это будет необходимо перед использованием большого количества процессоров, что станет полезным для массы потенциальных пользователей HPC.

Отказ – это опция

Говорят, что последние слова Будды были: "Распад присущ всем комплексным вещам". И Будда не был даже системным администратором. Кластеры – комплексная вещь. Чем больше кластер, тем больше частей, которые могут распадаться. Программа, которая обычно использует более чем 1000 процессоров, будет подвергаться отказам компонентов в некоторых точках. В качестве гипотетического примера, если у вас есть 1000 узлов кластера с MTBF (Mean Time Between Failure) 10000 часов (1,1 года), что означает, что можно ожидать отказ очередного узла каждые десять часов. Учитывая, что отказоустойчивость фиксируется для наибольшего количества компьютерной техники, использование все большего и большего количества процессоров для вашей программы в конечном итоге становится убыточным.

В будущем, я надеюсь, кластеры будут иметь постоянное (и ожидаемое) количество отказов. Кроме того, затраты на увеличение отказоустойчивости, вероятно, будут огромными и адаптация к провалу будет простым решением.

Затем я должен спросить: "Как написать программу для оборудования, если вы знаете, что произойдет сбой в любой момент?". Ответ прост - программа будет выдерживать аппаратные сбои. Другими словами, программы должны стать отказоустойчивыми. И, что важно, я программист не должен был писать это в моей программе.

Динамическая масштабируемость

Один из способов сделать программу отказоустойчивой - сделать ее динамически масштабируемой. То есть, она может изменить количество процессоров, используемых ей на лету. Добавление отказоустойчивости означает переделать работу так, чтобы некоторые механизмы предназначались для динамического назначения процессоров. Динамическая масштабируемость, следовательно, - это то, что я хочу в моей программе. Идея довольно проста, я хочу одну программу, которая может работать на 10000 процессорах, а также на одном процессоре. Конечно, задача может быть нерешаема на одном процессоре. В конце концов, если большая проблема требует 10 000 процессоров в течение часа, то следует принимать по 1 процессору на 10 часов (при условии достаточности памяти). Я должен, однако, быть в состоянии выполнить небольшой набор данных на одном процессоре, а затем масштабировать их до максимального числа процессоров, что данная проблема размера.

Например, если я в состоянии разработать программу на своем ноутбуке и запустить ее на шестнадцати процессоров кластера и запустить его без каких-либо изменений. Если кластер работает под управлением других программ, в то же время и есть только четыре простаивающих процессоров, то моя программа должна начать использовать этих четырех процессоров. Как только и другие процессоры станут доступны она должна стремиться только к тому, чтобы добавить больше процессоров для увеличения производительности. На более позднем этапе, если я хочу запустить свою программу более чем на 1000 процессорах, я должен быть в состоянии управлять этой программой.

Больше не придется стоять в очереди

Так как моя программа в настоящее время динамически масштабируемая, я полагаю, у вас также. В этом случае наши программы должны быть в состоянии сотрудничать друг с другом. Если мы оба имеем программы для запуска в то же время мы должны совместно использовать ресурсы оптимальным образом. Во многих случаях нужно запланировать запуск и ждать очереди рабочих мест не будет необходимости, потому что программы будут управлять сами. Моя программа будет постоянно вести переговоры с другими запущенными программами, чтобы получить лучший набор ресурсов кластера. Например, моя программа может вести переговоры, чтобы ждать, пока другие работают, если она может получить эксклюзивный доступ к 100 процессорам на один час. Меня не волнует, каким образом программы это сделают, я просто хочу, чтобы они вели себя подобным образом, и я не хочу, чтобы писать такое поведение в моей программе.

Кроме того, в рамках этой динамической схеме не должно быть центрального пункта управления. Программы должны вести себя независимо и не должны полагаться на один ресурс. Действительно, в самих программах подразделы должны быть автономными, насколько возможно. Централизованное управление шестнадцатью процессорами представляется разумным, управление тысячью – является реальной проблемой.

И еще кое-что

Наконец, я хочу выразительный язык, свободный от каких-либо артефактов в связи с базовым аппаратным обеспечением. Я хочу быть как можно ближе к приложению Я хочу кодировать насколько это возможно. Мыслить в "моей проблеме пространства", где я хочу жить. Что касается управления памятью, количество процессоров, и другие подобные детали уводит меня от предметной области. Короче говоря, это мой список пожеланий: отказоустойчивость, динамическая масштабируемость, кооперативность и выразительность. Простое желание, но трудная задача. Понимая, что я редко получают то, что я хочу, я поставил мои ожидания высоко, так, может быть, проще, может быть, я получу то, что мне нужно. Как мы собираемся достичь этой высоты ? Я думал, вы никогда не спросите. У меня есть несколько идей, но, во-первых, позвольте остановиться на некоторых вопросах, которые всегда, кажется, возникают, когда я говорю на эту тему.

А как насчет MPI ?

Позвольте мне пояснить. MPI (Message Passing Interface) и PVM (Parallel Virtual Machine), впрочем, прекрасные идеи. Они позволили мне и многим другим, используя коллекции процессоров, добиться многого. Будьте уверены, передачи сообщений не затмили новые технологии "программирования" в ближайшее время. В самом деле, по всей вероятности, быть в центре самых параллельных приложений в будущем, потому что вы не можете иметь параллельных вычислений без связи. Как важно, что MPI состоит в мире высокопроизводительных вычислений, оно представляет собой барьер на пути экспертов в предметной области. Это означает, что в программирование в MPI слишком много инвестировал Джо Сикспак. Это требует не только изменения кода, тестирования и отладки труднее, и возможные основные переписывания кода может быть необходимым. Конечно, результат может быть впечатляющим, но стоимость разработки не оправдает результат.

Даже если нам удается производить армии программистов MPI, есть еще один более тонкий вопрос, который необходимо решать. Как написано, большинство параллельных программ не может дать гарантии эффективного выполнения на каждом компьютере. Не существует никакой гарантии, что, когда я поставлю свои MPI / Pthreads / OpenMP программы на другой компьютер, он будет работать оптимально. Обсуждение этой темы выходит за рамки данной статьи, но позвольте мне сказать, что каждый кластер или SMP машина имеет уникальное соотношение вычислений для общения. Это соотношение определяет эффективность и должны учитываться при принятии решений о распараллеливании. Для некоторых приложений, таких как рендеринг, это отношение имеет большое значение, в других он может сделать огромную разницу в производительности и определить, как выделения и группировки кода. К сожалению, ваши нарезки и перетасовки могут хорошо работать в одной системе, но нет никакой гарантии, что будет хорошо работать на всех системах.

MPI часто называют машинным кодом для параллельных компьютеров. Я бы с этим согласился. Это портативный, мощный, и, к сожалению, на мой взгляд, слишком сложно для повседневного программирования. На мой взгляд, параллельные вычисления утопия, MPI и других подобных методов.

Абстрактное искусство

Восхождение выше потолка MPI не пройдет без затрат. Подобно тому, как происходит потеря возможной производительности при переходе от сборки языка C, будет потеря эффективности при программировании без явного сообщения. Этот термин часто используется как "более высокий уровень абстракции". Причины в языках высокого уровня настолько популярны, потому что они обеспечивают высокий уровень абстракции выше оборудования. Программисты приблизились к их применению и стали дальше от компьютера.

В моей давно забытой статье, я сделал так, что в первые дни вычислительных систем была огромная дискуссия относительно использования нового языка, называемый Фортран, а не ассемблера (машинного кода). Да, в те темные первые дни, не было Perl или Python и новые переводы формул стали прорывной идеей, поскольку она абстрагируется от некоторых из машины, и пусть ученые не являются программистами, им было очень легко понять эти формулы.