Раздел 7
Законы Амдала и Густафсона

В данном разделе вводятся новые понятия – уровни и метрики параллелизма, степень и профиль параллелизма, ускорение и эффективность. Изучается зависимость ускорения от характеристик алгоритма и числа процессоров – формулируется закон Амдала. Обуславливаются практические ограничения ускорения: алгоритмические издержки, дисбаланс загрузки процессоров, коммуникационные издержки. Рассматривается новый взгляд на закон Амдала – закон Густафсона.

Слайд 3

Распараллеливание операций – перспективный путь повышения производительности вычислений. Согласно закону Мура число транзисторов экспоненциально растёт, что позволяет в настоящее время включать в состав CPU большое количество исполнительных устройств самого разного назначения. Миновали времена, когда функционирование ЭВМ подчинялось принципам фон Неймана. Уже полвека назад медленные периферийные устройства работали асинхронно по отношению к центральному процессору.

В 70-е годы стал активно применяться принцип конвейеризации вычислений. Сейчас конвейер P4 состоит из 20 ступеней. Такое распараллеливание на микроуровне – первый шаг на пути эволюции процессоров. Микроуровневый параллелизм свойственен не только процессорам общего назначения (GPP), но и процессорам специального назначения (ASP, DSP). На принципах конвейеризации базируются и внешние устройства. Например, динамическая память (организация чередования банков) или внешняя память (организация RAID).

Но число транзисторов на чипе росло. Использование микроуровневого параллелизма позволяло лишь уменьшать CPI, так как миллионы транзисторов при выполнении одиночной инструкции простаивали. На следующем этапе эволюции в 80-е годы стали использовать параллелизм уровня команд посредством размещения в CPU сразу нескольких конвейеров. Такие суперскалярные CPU позволяли достигать CPI<1. Параллелизм уровня инструкций породил неупорядоченную модель обработки, динамическое планирование, станции резервации и т.д. От CPI перешли к IPC. Но ILP ограничен алгоритмом исполняемой программы. Кроме того, при увеличении количества ALU сложность оборудования экспоненциально растёт, увеличивается количество горизонтальных и вертикальных потерь в слотах выдачи. Параллелизм уровня инструкций исчерпал себя, а тенденции Мура позволили процессоростроителям осваивать более высокие уровни параллелизма. Современные методики повышения ILP основаны на использовании процессоров класса SIMD. Это векторное процессирование, матричные процессоры, архитектура VLIW. Этим вопросам будут посвящены отдельные лекции.

Параллелизм уровня потоков и уровня заданий применяется в процессорах класса MIMD. Многопоточные процессора позволяют снижать вертикальные потери в слотах выдачи, а Simultaneous Multithreading процессора (SMT) – как вертикальные, так и горизонтальные потери. Закон Мура обусловил также выпуск многоядерных процессоров (CMP). Современные топовые вычислители – это мультикомпьютерные мультипроцессорные системы. Этим вопросам будут посвящены отдельные лекции.

Параллелизм всех уровней свойственен не только процессорам общего назначения (GPP), но и процессорам специального назначения (ASP, DSP).

Иногда классифицируют параллелизм по степени гранулярности как отношение объёма вычислений к объёму коммуникаций. Различают мелкозернистый, среднезернистый и крупнозернистый параллелизм. Мелкозернистый параллелизм обеспечивает сам CPU, но

компилятор может и должен ему помочь для обеспечения большего IPC. Среднезернистый параллелизм – прерогатива программиста, которому необходимо разрабатывать многопоточные алгоритмы. Здесь роль компилятора заключается в выборе оптимальной последовательности инструкций (с большим IPC) посредством различных методик (например, символическое разворачивание циклов). Крупнозернистый параллелизм обеспечивает ОС.

Слайд 4

Законы Амдала и Густафсона, к которым мы приближаемся, ориентированы на крупнозернистый параллелизм. Допустим, имеется в наличии вычислительная система с неограниченными вычислительными ресурсами (в которой количество процессоров неограниченно). Как поведут себя программы в этой системе? Ускорятся ли они в неограниченное количество раз? Разумеется, нет! Скорость работы простейшей однопоточной программы не изменится, так как она не использует параллелизм высокого уровня. Её вычислительная нагрузка ляжет исключительно на один из процессоров. В отличие от микроуровневого параллелизма, который обеспечивается самим процессором, или в отличие от параллелизма уровня инструкций, где участия программиста не требуется (за исключением SIMD), привлечение к вычислениям нескольких процессоров требует существенной модификации алгоритма. Фактически требование дальнейшего повышения производительности вычислений вынуждает применять новые методики программирования, такие как векторное процессирование (например, SSE), многопоточное программирование (например, MPI) и т.д. Отметим, что классические языки программирования высокого уровня (такие, как C++) ориентированы исключительно на класс SISD.

Итак, программист изменил алгоритм и создал многопоточный код (например, при помощи потоков POSIX). Приведёт ли это к неограниченному росту производительности? Введём в рассмотрение степень параллелизма программы – D(t) – число процессоров, участвующих в исполнении программы в момент времени t. При старте программы D(0)=1. Далее программа может создавать независимые потоки и передавать им часть своей нагрузки. Изучив поведение алгоритма на неограниченной машине, мы получим график D(t) – профиль параллелизма программы. Однако степень параллелизма зависит не только от используемого алгоритма программы, но и от эффективности компиляции и доступных ресурсов при исполнении. Реальное значение D(t) будет иным.

Введём в рассмотрение T(n) – время исполнения программы на n процессорах. $T(n)<T(1)$, если параллельная версия алгоритма эффективна. $T(n)>T(1)$, если накладные расходы (издержки) реализации параллельной версии алгоритма чрезмерно велики. Внимание: за T(1) взято не время выполнения многопоточной программы на одном процессоре, а время выполнения однопоточной программы.

Слайд 5

Тогда ускорение за счёт параллельного выполнения составит $S(n) = T(1) / T(n)$.

С экономической точки зрения интерес представляет ускорение в пересчёте на один процессор – эффективность системы из n процессоров $E(n) = S(n) / n$.

В идеальном случае ускорение линейно от n. Такой алгоритм обладает свойством масштабируемости (возможностью ускорения вычислений пропорционально числу процессоров). Но на практике масштабируемый алгоритм имеет несколько худшие показатели из-за накладных расходов на поддержку многопроцессорных вычислений. Кроме того, параметр n масштабируемого алгоритма должен быть согласован с количеством процессоров в вычислительной системе. Необходимо также учитывать и текущую загруженность процессоров другими задачами.

Особое внимание заслуживает случай S(n) > n. Изначально может показаться, что такого не может быть. Однако примеры такого успешного распараллеливания есть! Вопрос лишь в том, как квалифицированно разбить исходную задачу на подзадачи. Требования исходной задачи могут превосходить возможности эксплуатируемого процессора по любому из его ресурсов (чаще всего это кеши различных уровней, буфер BTB, буфер TLB). После разбиения на один процессор попадает задача меньшего объёма, и, соответственно, требования к объёму ресурсов процессора сокращаются. При преодолении таких порогов возникает суперлинейное ускорение. Внимание: организация кода/данных должна обеспечивать максимальную степень локальности кода/данных (иначе ни линейного, ни тем более суперлинейного ускорения достичь не удастся).

Слайд 6

Сформулируем закон Амдала. Любая многопоточная программа содержит последовательную часть. Это ввод/вывод, менеджмент потоков, точки синхронизации и т.п. Обозначим долю последовательной части за f. Тогда доля параллельной части будет 1-f. Амдал в 1967 году рассмотрел ускорение такой программы на n процессорах, исходя из предположения линейного ускорения параллельной части, и получил $n/(1+(n-1)\times f)$. При неограниченном числе процессоров ускорение составит всего лишь 1/f.

Засада! Если, например, доля последовательной части 20%, то теоретически невозможно получить ускорение вычислений более чем в пять раз! Таким образом, превалирующую роль играет доля f, а вовсе не число процессоров!

Ещё пример. При переходе с однопроцессорной машины на четырёхпроцессорную возможно ли ускорение вычислений в два раза? Правильный ответ – не всегда! И опять всё упирается в долю f.

Слайд 7

По графикам S(f) можно увидеть, что при f большем 50% ни о каком существенном ускорении говорить не приходится. Только если доля f мала, становится экономически целесообразным многократное увеличение числа процессоров.

Слайд 8

Более того, имея такую грустную картину в теоретическом плане, на практике не избежать накладных расходов на поддержку многопоточных вычислений.

Это и алгоритмические издержки (менеджмент потоков и т.п.).

Это и коммуникационные издержки на передачу информации между потоками.

Это и издержки в виде дисбаланса загрузки процессоров. Действительно, даже если удалось разбить исходную задачу на равные по сложности подзадачи, время их выполнения может существенно различаться по самым разным причинам (от конфликтов в конвейере до планировщика ОС). В точках синхронизации (хотя бы в точке завершения всей задачи) потоки вынуждены ожидать самых слабых, что приводит к простою значительной части процессоров.

На этой пессимистичной ноте мысли о суициде неприемлемы, ибо…

Слайд 9

Оптимистичный взгляд на закон Амдала даёт закон Густафсона-Барсиса. Вместо вопроса об ускорении на n процессорах рассмотрим вопрос о замедлении вычислений при переходе на один процессор. Аналогично за f примем долю последовательной части программы. Тогда получим закон масштабируемого ускорения, и $S(n)=n+(1-n)\times f$.

Слайд 10

Теперь графики S(f) демонстрируют совершенно иную картину: линейное ускорение в зависимости от числа процессоров. Т.е. законы Амдала и Густафсона в идентичных условиях дают различные значения ускорения. Где же ошибка? Каковы области применения этих законов?

Густафсон заметил, что, работая на многопроцессорных системах, пользователи склонны к изменению своего поведения. Теперь снижение общего времени исполнения программы уступает объёму решаемой задачи. Такое изменение цели обусловливает переход от закона Амдала к закону Густафсона.

Например, на 100 процессорах программа выполняется 20 минут. При переходе на систему с 1000 процессорами можно достичь времени исполнения порядка 2 минут. Однако для получения большей точности решения имеет смысл увеличить на порядок объём решаемой задачи (например, решить систему уравнений в частных производных на более тонкой сетке). Т.е. при сохранении общего времени исполнения пользователи стремятся получить более точный результат.

Увеличение объёма решаемой задачи приводит к увеличению доли параллельной части, так как последовательная часть (ввод/вывод, менеджмент потоков, точки синхронизации и т.п.) не изменяется. Таким образом, уменьшение доли f приводит к перспективным значениям ускорения.