(без)перспективные области применения многопроцессорных машин

крис касперски, no-email

тактовую частоту невозможно наращивать бесконечно и предел насыщения уже близок. производители процессоров просекли ситуацию и вместо того, чтобы радовать нас новыми мегагерцами увеличивают количество ядер, что равносильно приобретению многопроцессорной машины, в которых, кстати говоря, потребили видят чуть ли не единственный резерв повышения производительности, но увы… очень часто их ожидания не оправдываются — установка новых процессоров не увеличивает быстродействия сервера/рабочей станции и вложенные средства не окупаются. почему? попробуем разобраться!

введение

На интуитивном уровне — два процессора лучше одного, а четыре лучше двух, но результаты тестов показывают обратное. Зачастую переход от однопроцессорной машины к многопроцессорной не только не увеличивает, но даже уменьшает производительность (причем весьма значительно). Это кажется невероятным, но таковы факты, а факты — штука упрямая.

Расчет оптимального количества процессоров в вычислительной системе — сложная штука, требующая не только владения математическим аппаратом на надлежащем уровне, но и учета особенной архитектуры программно-аппаратной среды, которая в общем случае недоступна для анализа. Небольшое количество приложений с отрытыми исходными текстами — это лишь капля в море проприетарного программного обеспечения. Мы знаем лишь то, что разработчики оптимизируют алгоритмы в стремлении добиться максимальной производительности, но насколько это им удается — это вопрос. Если поставщик не дает внятных рекомендаций: сколько требуется процессоров для обработки обозначенного количества запросов в единицу времени, нам остается действовать методом научного тыка, продвигаясь во тьме наугад.

К счастью, оценить приблизительный коэффициент кол-во процессоров/производительность можно и без приобретения дополнительного оборудования.

Рисунок 1 фото автора

планирование, планирование и еще раз планирование

Программа (в классическом понимании этого слова) представляет собой последовательность машинных команд (инструкций), выполняемых процессором в порядке их расположения. И хотя современные процессоры для увеличения производительности способны выполнять более одной инструкции одновременно или даже переупорядочивать команды, чтобы более медленные инструкции не тормозили весь поток управления. Грубо говоря, если уподобить команды автомашинам, то их переупорядочивание равносильно обгону грузовика Мерседесом.

Хотите конкретный пример — пожалуйста:

```
LODSD                   ; загрузка ячейки из оперативной памяти в регистр EAX
SUB     ECX, EDX        ; вычитание содержимого регистра EDX из регистра ECX
```

Листинг 1 пример кода, который можно ускорить путем переупорядочивания команд

Команда LODSD, выполняющая загрузку ячейки памяти, может выполняться долго. Очень долго. Если требуемая ячейка отсутствует в кэше и находится в оперативной памяти, которая в этом момент закрыта на регенерацию, это будет стоить процессору _тысячи_ тактов, а то и больше. А вот команда, следующая за ней, выполняется всего за один такт. Так зачем же процессор будет простаивать как дурак, если он может преспокойно продолжить поток управления?!

Сложность в том, что если на пути процессора встретится команда, обращающая к загружаемой ячейке, то процессор будет вынужден "обогнать" и ее (ведь ячейка еще не загружена!), а тут вдруг на трассу вылетает встречная — инструкция, оперирующая с результатами вычислений "обогнанной" команды. И что должен делать процессор?! Правильно — разбить поток управления на отдельные команды, проанализировать имеющиеся зависимости по данным и выполнять код так, чтобы сократить время простоя до минимума.

Однако, анализ потока управления — довольно сложная задача и "окно" анализа составляет порядка десятка инструкций. То есть, несмотря на все усилия разработчиков процессоров, программа продолжает выполняться последовательно и степень параллелизма крайне невелика (вообще говоря, переупорядочиванием инструкций занимаются не только процессоры, но и компиляторы).

Добавление второго (четвертого, восьмого) процессора не увеличит скорость выполнения программы, поскольку… ну это как три человека сажают дерево. Один копает яму, второй держит дерево, а третий ее засыпает. Попытка распараллелить эти операции приведет к тому, что яма вообще не будет выкопана! А вот если мы сажаем несколько деревьев, то производительность труда окажется прямо пропорциональной количеству землекопов, при условии, конечно, что этот труд организован надлежащим образом.

Таким образом, эффективное использование нескольких процессоров все-таки возможно, но для этого программный код должен быть _изначально_ сконструирован в расчете на параллелизм. Практически все современные операционные системы поддерживают концепцию _потоков_ — самостоятельных сущностей внутри программы, выполняющихся параллельно друг с другом. Рассмотрим вполне типичный сценарий при котором один поток занят вычислениями, а другой выводом результатов на экран.

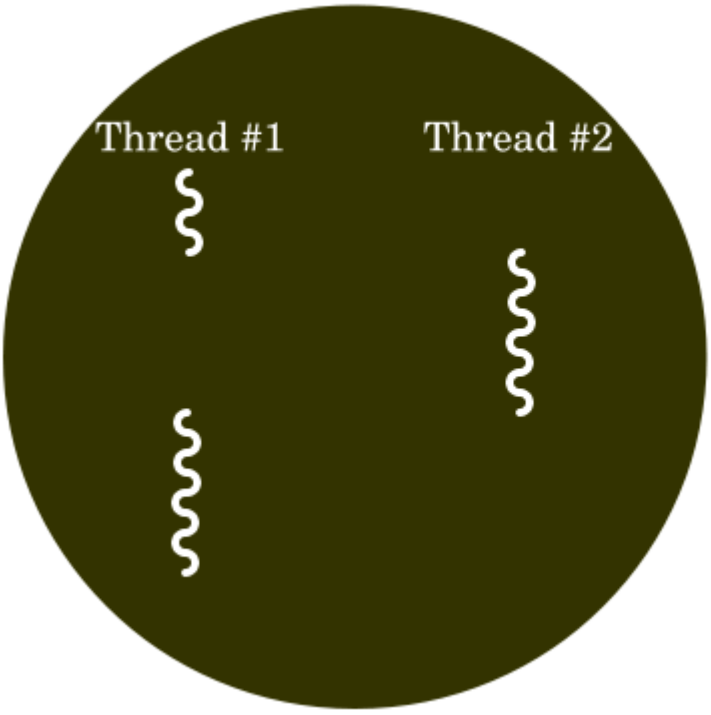

Рисунок 2 потоки как они есть

Специальный компонент системы, именуемый планировщиком, отводит каждому из потоков определенное время (называемое квантом) и по истечении этого кванта принудительно отбирает у потока управление, передавая его другому, наиболее нуждающемуся потоку (иногда говорят, что планировщик ставит потоки в очередь, но это уж слишком большое упрощение). Длительность кванта и алгоритм выбора потока, больше других нуждающегося в управлении, зависит от операционной системы и постоянно совершенствуется разработчиками. В Windows длительность кванта составляет порядка 100 мс, и если у нас имеется всего один процессор и два потока и эти потоки полностью расходуют процессорное время, то они получают управления каждые 100 мс, хотя, поток, отвечающий за вывод на экран, большую часть времени проводит в спячке, возвращая остаток кванта вычислительному потоку, хотят тут все зависит от рода вычислений. Если мы, скажем, вычисляем сумму ячеек таблицы, отображая ее в виде вращающейся трехмерной диаграммы, то тут основная нагрузка ложится уже на второй поток.

Суть в том, что планировка процессорного времени осуществляется на уровне потоков (или процессов, если операционная система не поддерживает потоков). Асимметричные операционные системы (не получившие большого применения) закрепляют один процессор за системной, а остальные равномерно распределяют между имеющимися потоками. Операционные системы с симметричной многопроцессорной обработкой (к которым относятся Windows, Linux и BSD) не делают между системными и несистемными потоками никаких различий, предоставляя потоку процессор как только подойдет его очередь.

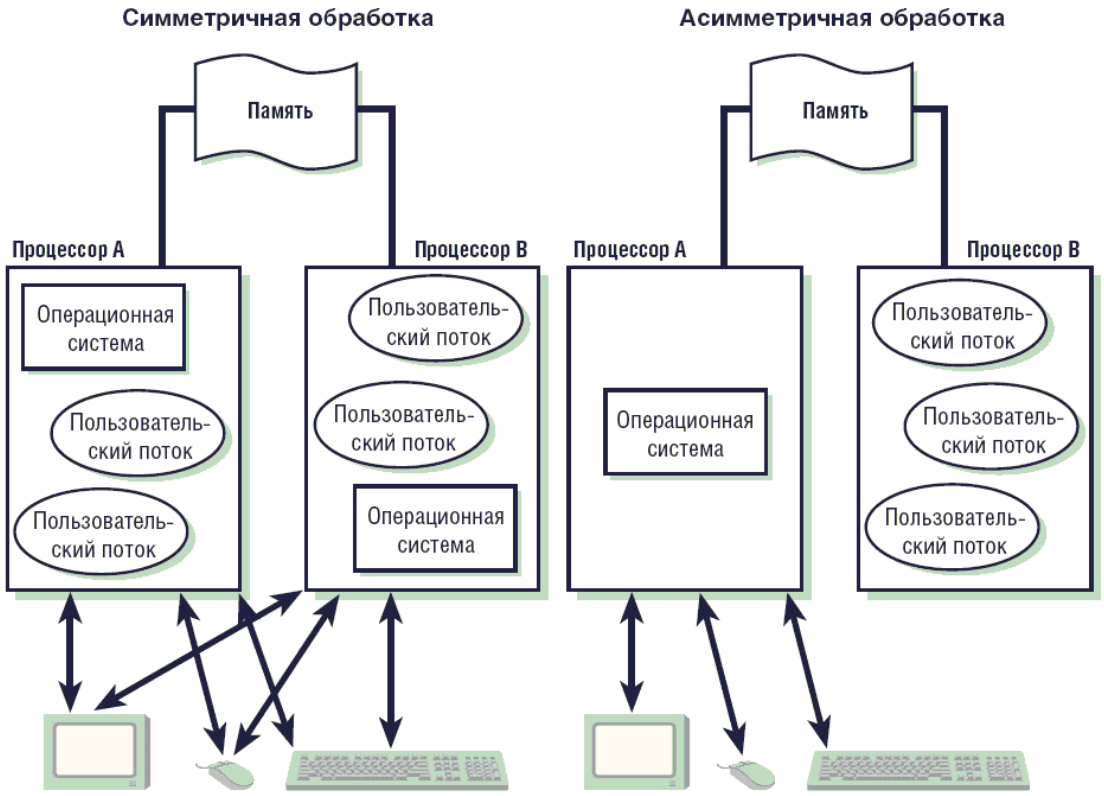

Рисунок 3 операционные системы с симметричной и асимметричной обработкой потоков

А теперь попробуем ответить на вопрос: как изменится производительность выше обозначенной программы, если ее запустить на двухпроцессорной системе? Если программа сконструирована правильно и 90% нагрузки ложится на вычислительный поток, то производительность сильно не возрастет, т. к. в каждый момент времени вычислительный поток может выполняться только на одном процессоре! Скажем больше, при добавлении еще одного процессора производительность даже _упадает_ причем весьма ощутимо.

Во-первых, многопроцессорные ядра операционных систем за счет более сложной организации вынуждены платить дань накладных расходов. Во-вторых, каждый процессор имеем свой собственный кэш и если вычислительный поток попадает то на один, то на другой процессор, падение скорости из-за необходимости "перегона" данных туда-и-обратно окажутся просто драматическими.

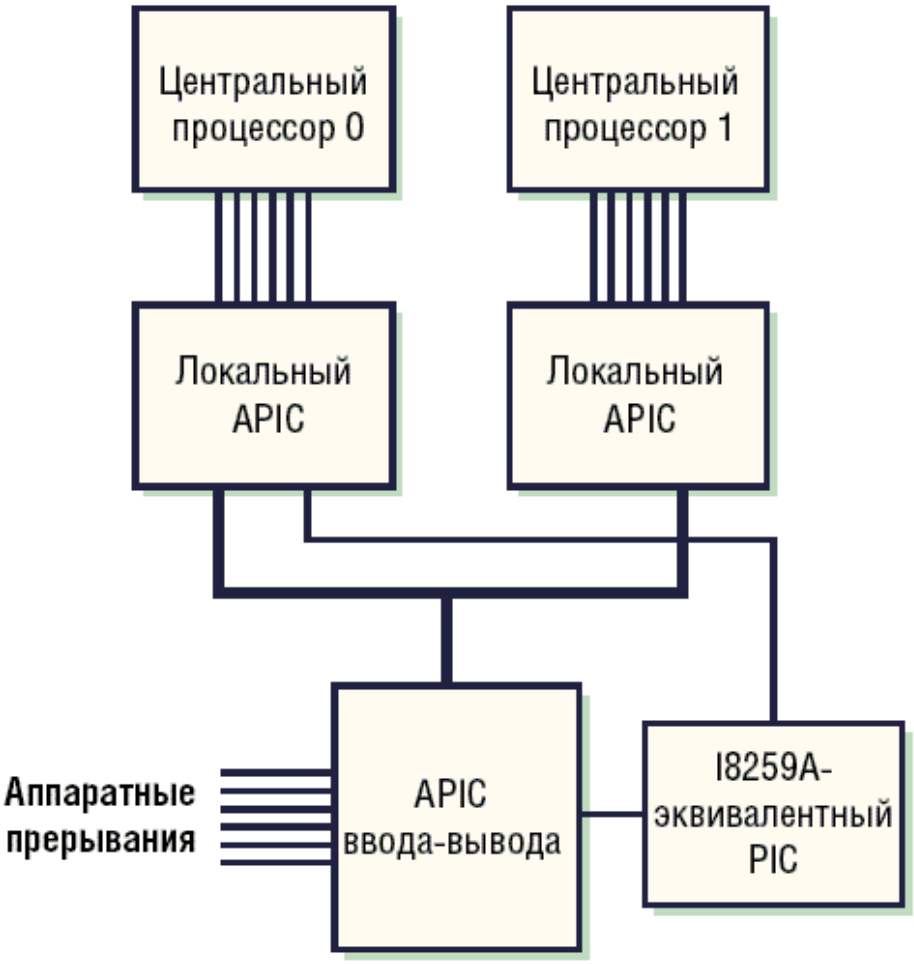

Рисунок 4 упрощения схема соединения двух процессоров

Начиная с Server 2003, в Windows появились API-функции, позволяющие "закреплять" поток за своим процессором, но… для этого программист их должен использовать _явным_ образом, что происходит далеко не везде и не всегда, а если и происходит, то вносит в работу планировщика определенную сумятицу, что опять-таки снижает эффективность параллелизма.

Следовательно, определенному типу приложений многопроцессорные системы просто-напросто противопоказаны!!! И таких приложений — большинство! Достаточно запустить "Диспетчер Задач", а еще лучше Process Explorer Марка Руссиновича и посмотреть сколько потоков у подопытного приложения и сколько какой из них "съедает" процессорного времени. Потоков чаще всего будет намного больше одного, но… все это вспомогательные потоки, а основная нагрузка ложится на базовый поток, в котором программа проводит свыше 90% своего времени.

Ладно, оставим рабочие станции и возьмемся за сервера. Отличная идея обрабатывать каждое подключение в отдельном потоке! Тогда между производительностью и числом процессоров установится линейная зависимость, что следует из самых общих соображений. Идея, конечно хорошая, но давайте приложим к ней математический аппарат. Длительность одного кванта, как уже говорилось, составляет порядка 100 мс. Допустим, что с сервером установлено 100 соединений (такой небольшой домашний сервер, да?). Считаем: если каждое соединение обрабатывается "своим" потоком, то на однопроцессорной машине оно получает управление один раз за 10 сек. Не миллисекунд, а именно секунд!!! Далеко на таком сервере не уедешь… Вот программистам и приходится использовать так называемую "легкую многозадачность" или эмулировать потоки внутри потоков, которые не планируется, то есть обрабатываются системой как "монолитный" поток.

Естественно, если сервер написан с оглядкой на многопроцессорные машины, то он создаст более одного "каркасного" потока, равномерно распределяя нагрузку между всеми имеющимися процессорами и зависимость производительность — количество процессоров при такой схеме действительно должна быть близка к линейной. Должна, но, увы, не обязана. Ведь

помимо процессоров существует еще и дисковая подсистема, и оперативная память, и многие другие "узкие" места. Если это простой WEB или FTP сервер, то основная нагрузка ложится на дисковую подсистему (плюс оперативную память, используемую в качестве кэша для часто запрашиваемых данных). RAID-контроллер увеличит производительность в гораздо большей степени, чем добавление процессоров, да и в плане финансов последний обойдется дешевле.

Другими словами, процессоры нужны там и только там, где требуются _вычислительные_ ресурсы и где все прочие подсистемы компьютера работают с запасом, а не прогибаются под нагрузкой. Ладно, оставим WEB/FTP сервера, оставим базы данных и прочие задачи, главным образом завязанные на дисковую подсистему, и займемся нелинейным видеомонтажам. Сжатие аудио/видео данных требует огромных вычислительных ресурсов и если тактовой частоты для достижения заданной производительности недостаточно, быть может, стоит увеличить количество процессоров?!

Сразу же возникает вопрос: а "знает" ли используемая программа, что ее будут использовать на многопроцессорной машине? Сумеет ли она распараллелить сжатие между несколькими потоками и насколько это распараллеливание окажется эффективным?! Наконец, как насчет пропускной способности дисковой подсистемы и оперативной памяти?! Теоретически линейная зависимость вполне достижима и лучшие из программ вплотную приблизились к этому пределу, но в большинстве случаев коэффициент составляет 0,5 или же производительность растет пропорционально квадратному корню от количества процессоров.

К сожалению, увлеченные гонкой за мегагерцами, программисты оказались совсем не готовы к параллельным вычислениям и большинство программного обеспечения, имеющегося на рынке, встречает многопроцессорные системы без должного энтузиазма. С другой стороны, компьютер из электронно-вычислительной машины уже давно превратился в своеобразный микроконтроллер, управляющей вводом/выводом. Конечно, это сказано несколько утрировано, но… запрос к базе данных не требует вычислений, хотя процессор и выполняет при этом огромное количество арифметическо-логических операций, но это совсем не та "арифметика" для которой создавались вычислительные машины.

А вот криптография, биллинг, компьютерное моделирование и другие подобные им задачи на 90% состоят из вычислительной части и потому, казалось бы, очень чутко должны реагировать на количество процессоров в системе. Но… увы… программы, изначально создаваемые для ПК, сконструированы в расчете на один процессор и попытки "привить" им многопроцессорность на поздних стадиях проектирования редко приводят к успеху (не говоря уже о том, что некоторые задачи нельзя распараллелить в принципе, например, посадку одного-единственного дерева).

Прежде чем устанавливать новый процессор в систему необходимо спросить себя: а найдется ли для него работа? Сажать дерево можно и впятером и вдесятером, вот только в каждый момент времени будет работать только один человек (тот, что держит дерево — не в счет, он все равно, что спит), а остальным придется стоять в сторонке и курить.

от теории к практике

Теория — это, конечно, хорошо. ОК, мы поняли, что больше процессоров далеко не всегда означает лучше и необходимость перехода на многопроцессорную систему еще надо обосновать. А вот как ее обосновать?!

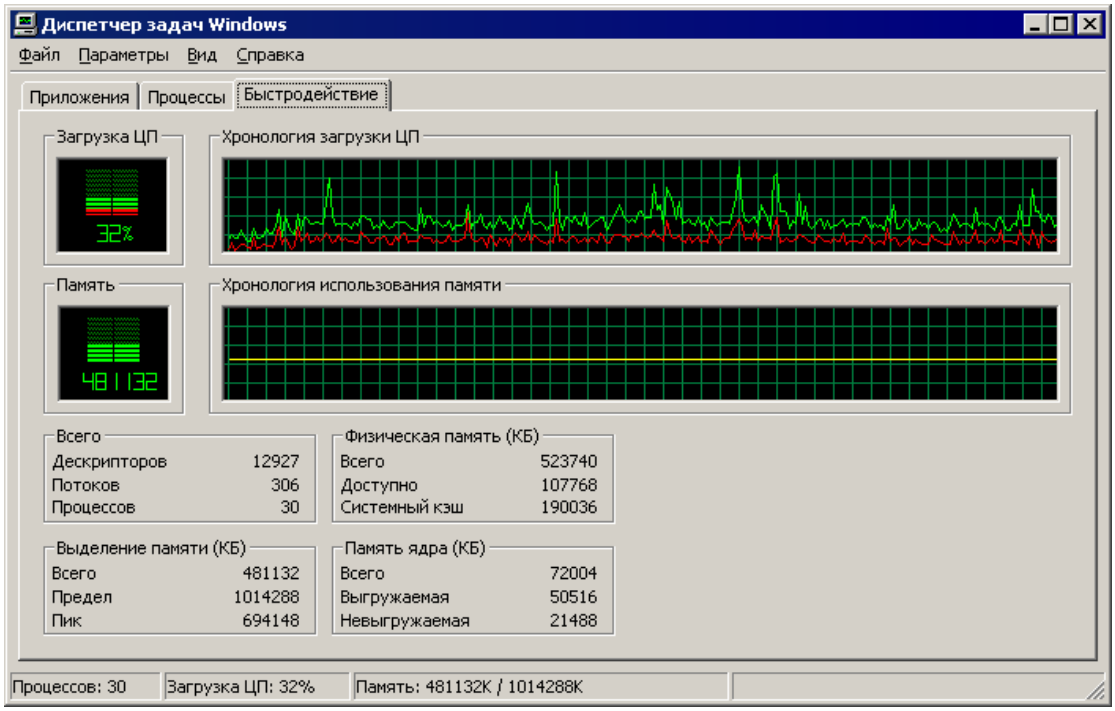

Рисунок 5 определение загрузки ЦП с помощью штатного "Диспетчера Задач"

Современные операционные системы поддерживают развитые механизмы счетчиков производительности (performance counters), анализ которых позволяет выявить узкие места, оказывающие наиболее радикальное влияние на быстродействие компьютера. В идеально сбалансированной системе узких мест нет и для увеличения производительности хотя бы на треть, необходимо обновлять _все_ ее компоненты. Разумеется, идеалы существуют только в теории, и на практике недостижимы. Чем плоха несбалансированная система? А тем, что часть компонентов компьютера работает далеко не на пределе своих возможностей, а за эти возможности, между прочим, деньги заплачены!

Проблема, однако, в том, что показания счетчиков производительности невозможно выразить в естественных, интуитивно понятных единицах, и чтобы правильно их проинтерпретировать необходимо отчетливо представлять себе внутренний мир операционной системы, довольно поверхностно описанный в доступной литературе и, как следствие, окруженный кучей мифов, сплетен, вымыслов и заблуждений.

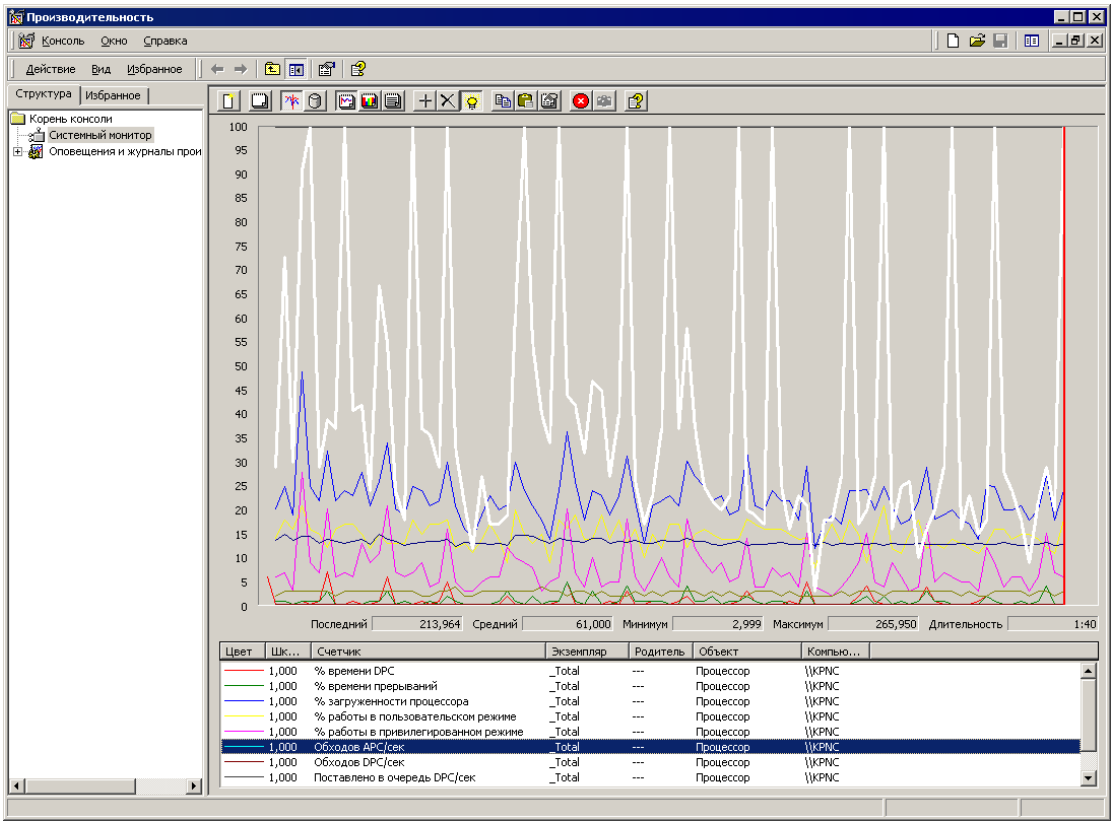

Рисунок 6 определение загрузки ЦП с помощью "Системного Монитора"

Взять хотя бы показать "Загрузка ЦП" в "Диспетчере Задач" (соответствующий счетчику производительности "Processor: % Processor Time:" Системного Монитора. Заблуждение первое (и невероятно широко распространенное): если загрузка ЦП далека от 100%, то все ОК и установка новых процессоров не требуется. На самом деле, данный счетчик производительности измеряет не загрузку ЦП как таковую, а готовность системного планировщика предоставить процессорное время потоку по первому требованию. Это очень грубый показатель, а методы его измерения вообще таковы, что вызывают шевеление волос в разных местах. Упрощенно все происходит приблизительно так. Имеется рабочий, который работает. Или не работает. И специальный инспектор через регулярные промежутки времени (например, каждый час) приходит и поверяет чем тот в данный момент занимается. Если рабочий вкалывает как пара Карло, то ему ставится "зачот" за весь отчетный период и, соответственно, наоборот.

Вообразим вполне вероятную ситуацию, при которой рабочий устраивает себе пятиминутный перекур каждый час. Как нетрудно рассчитать, его "загрузка" составит ~90%, но!!! Если перекуры совпадут с приходом инспектора, мы получим… нулевую загрузку!!! Шутки в сторону, господа! При желании можно написать программу, потребляющую свыше 90% времени ЦП, но "Диспетчер Задач" (а вместе с ним и "Системный Монитор") будут осцилировать в переделах абсолютного нуля. Для этого достаточно отдавать остаток кванта времени за несколько миллисекунд до его истечения. Планировщик, обнаружив, что поток не исполняется, ошибочно пропишет ему ноль в графе "использование процессорного времени".

Самое интересное, что чаще всего это возникает не по злому умыслу (кому-то захотелось украсть процессорное время), а по доброте душевной. Следуя этикету многозадачности, потоки должны делиться квантами времени, потому что, как уже говорилось, если они этого не будут делать, то система ляжет. На моем домашнем компьютере в настоящий момент исполняется 296 потоков. Вспомнив, что длительность кванта составляет 100 мс, мы получаем цифру в ~30 сек. Именно столько времени должен находится в очереди каждый поток, ожидая своей порции машинного времени. Но потоки делятся остатками выделенных им квантов и потому система работает без тормозов с загрузкой ЦП в районе 20%-30%. Однако, это вовсе не означает, что ресурсов ЦП достаточно для решения поставленных задач и что переход на многопроцессорную машину не увеличит производительность. Может быть увеличит, а быть может и нет, это от кучи самых разных обстоятельств зависит!!!

А вот другое заблуждение. 100% загрузка ЦП — сигнал о том, что нам нужно увеличить количество процессоров или повысить их тактовую частоту. Возвращаясь к нашей аналогии с рабочим и инспектором, напишем несложную программку, один из потоков которой в ожидании какого-то события (например, завершения передачи данных по сети) вместо того, чтобы вогнать себя в сон (следуя предписаниям Microsoft), будет мотать холостой цикл. И что же?! Цикл может быть и холостой, а загружает процессор (на однопроцессорной машине) на все 100%!!! Естественно, увеличение тактовой частоты процессора хоть и увеличит скорость выполнения холостого цикла, производительности не поднимет. А вот переход на двухпроцессорную машину сразу же сбросит загрузку ЦП со 100% до 50%, при условии отсутствия других потоков в системе, конечно. А что если у нас имеется два потока, один из которых выполняет полезные вычисления, а другой — просто наматывает цикл на кардан? На однопроцессорной машине они получают равные порции машинного времени — каждый по 50%, что в совокупности дает 100% загрузку ЦП. Переход на двухпроцессорную машину увеличит "паек" каждого из потоков в два раза и производительность пропорционально возрастет, а вот 100% загрузка ЦП как была так и останется. Естественно, на четырехпроцессорной машине, загрузка окажется всего 25%, но производительность — та же, что и на двухпроцессорной. Вот такая арифметика.

Так как же все-таки интерпретировать показания счетчиков производительности?! Как определить — достаточно ли у нас мощности ЦП или нужно наращивать ее в срочном порядке?! Ответ обескураживающий — никак. Если уж совсем невмоготу, попробуйте обратить внимание на счетчик "System: Processor Queue Length", отображающий длину очереди потоков, простаивающих в ожидании процессора. Естественно, чем длиннее очередь — тем ниже производительность. Если среднестатистическая длина очереди превышает 10 потоков, имеет смысл задуматься о добавлении новых процессоров или увеличить таковую частоту, тогда очередь потоков будет продвигаться быстрее, в результате чего ее длина сократиться, однако, никаких гарантий, что производительность действительно возрастет у нас нет. Увы!

заключение

Кажущаяся "крутость" многопроцессорных систем и благородный ореол, витающий вокруг них, в первую очередь объясняется высокой ценой последний, а, следовательно, малой распространенностью. До появления многоядерных процессоров очень мало кто реально пытался оценить какой выигрыш дает перевод сервера/рабочей станции на многопроцессорную платформу.

Теперь же, в результате падения цен на вычислению технику, многопроцессорные системы перестали быть экзотикой и вышли в люди. Чем закончилась эта встреча? Помните, когда Linux был уделом горстки хакеров, среди пользователей ходили слухи о том какая это надежная и устойчивая система по сравнению с ненавистной Windows? Но стоило Linux'у дорасти (вот именно дорасти!) до минимально приемлемого потребительского уровня, как все увидели, что он не только не падает, но еще и зверски тормозит и при прочих равных Windows 2000 или XP требуют куда меньше памяти и мегагерц.

Вот так же и с многопроцессорными системами. Поставив их на свой рабочий стол многие вообще не заметили никакой существенной разницы, что не покажется удивительным, если принять во внимание все, сказанное выше.

Подведем итог: многопроцессорные системы экономически оправданы лишь при работы с программами, в которых параллелизм заложен изначально и рекомендуемое количество процессоров _явно_ оговорено в технической документации, от которой и стоит плясать, а вовсе не от независимых обзоров, которые явно не дружат ни с логикой, ни с основами метрологии.