Марк Гюнтер, *Wind River*

Оптимизированное ПО для многоядерных процессоров обеспечивает прорыв в пропускной способности сетей

В статье обсуждается роль, которую ПО многоядерных вычислительных систем играет в увеличении производительности сетей передачи данных, а также архитектурные принципы построения многоядерных систем обработки пакетов и их влияние на технико-экономические показатели.

Введение

Технологии многоядерных вычислений всё прочнее обосновываются в архитектуре современных сетевых устройств – устройства доступа, оконечные и базовые устройства всё чаще пользуются преимуществами, которые предлагают процессоры с двумя и более ядрами. Основные предпосылки применения многоядерных процессоров в сетевом оборудовании – более тесная интеграция функций и прирост производительности за счёт использования более совершенных методик обработки трафика. Один из основных акцентов в современной индустрии сетей передачи данных – это увеличение производительности оборудования с целью справиться с обработкой растущих объёмов трафика, генерируемого беспроводными и иными связанными вычислительными устройствами. Симметричная мультипроцессорность не позволяет в полной мере раскрыть потенциал многоядерных вычислений в сетевых приложениях; здесь требуется более широкий подход, охватывающий как многоядерных процессоры, так и операционные системы и сетевые протоколы.

В современной сетевой инфраструктуре результирующая удовлетворенность потребителя может напрямую зависеть от производительности приложений и их способности обрабатывать большие объёмы данных в единицу времени. Тем, насколько быстро производители оборудования находят пути оптимизации производительности, часто определяется успех или провал.

В настоящей статье обсуждается роль, которую ПО многоядерных вычислительных систем играет в увеличении производительности сетей. Современное разнообразие многоядерных процессоров предлагает множество ценных технологических возможностей, но использовать их в полной мере можно только с помощью специализированного ПО.

Эти вездесущие сети

IP-сети давно стали стандартом коммуникаций в практически любых системах, где есть микропроцессоры – от мощных маршрутизаторов, составляющих основу Интернет, до носимых мобильных устройств. Эти сети могут быть носителями любых данных – от сообщений электронной почты и веб-контента до потокового видео/аудио и информации LBS-сервисов. Преимущества универсального подключения к широкому спектру информационных служб высоко ценятся потребителями, что, в свою очередь, выражается в прибыли сервис-провайдеров и поставщиков оборудования (можно, впрочем, поспорить, что потребность в мультимедийном контенте была всегда и просто дожидалась момента, когда технологическая база дорастёт до нужного уровня). Объединение разнородных сервисов передачи данных требует применения в сетях более совершенных и более быстродействующих узловых устройств. Домашним шлюзам, например, теперь требуется предоставлять «коктейль» из доступа в Интернет, VoIP и потокового видео; а персональным

мобильным устройствам типа iPhone – всё вышеперечисленное, плюс ещё и в более миниатюрном формфакторе.

Кроме необходимости в расширении пропускной способности для передачи мультимедийного контента, сервис-провайдеры сталкиваются также с проблемой обслуживания достаточно большого числа абонентов, чтобы сохранить прибыльность. Это, в свою очередь, требует высокой пропускной способности транспорта и высокой производительности обработки пакетов от оконечных устройств. Чтобы решить эту проблему, одних широких каналов недостаточно; сервис-провайдерам приходится искать эффективные способы расширения абонентской ёмкости, не упуская из виду сопряжённые затраты. В современной мировой экономике маржинальная прибыль постоянно падает, а цикл разработки – укорачивается. Простое добавление ещё одного «лезвия» в существующую систему последнего поколения может не решить проблемы сроков, стоимости и функциональности одновременно. Корень проблемы лежит в оборудовании обработки пакетов: именно эффективная обработка пакетов создаёт почву для улучшения качества обслуживания, уменьшения «текучки» клиентов и роста прибыльности. Чтобы создавать высокопроизводительное, высокоинтегрированное и масштабируемое сетевое оборудование, необходим кардинально новый подход.

Своевременное появление многоядерных процессоров представило новую парадигму обработки сетевых пакетов. Большинство производителей кремния сейчас предлагает 2-, 4-, 8- и 16-ядерные процессоры, и они уже не ассоциируются исключительно с лабораторными серверами и рабочими станциями проектировщиков – они будут играть критическую роль в сетевом оборудовании нового поколения и эволюции общественных и частных сетей. Чтобы понять, как многоядерные вычисления могут повысить производительность сетей, давайте сначала рассмотрим общие преимущества применения многоядерных процессоров.

Выпрямить кривую производительности?

За последние несколько десятилетий все мы наглядно убедились, что число транзисторов на кристалле удваивается каждые два года (явление, известное как закон Мура). Однако кривая реальной производительности процессоров при этом постепенно начинает «заваливаться», так как в последние годы процессоры оказались неспособны использовать возросшее число транзисторов из-за ограничений по рассеиваемой мощности.

Многоядерный подход предлагает новую парадигму. Использование нескольких ядер параллельно позволяет увеличить вычислительную мощность до необходимого уровня; при этом энергопотребление, рассеиваемая мощность и стоимость получаются ниже, чем у аналогичных одноядерных процессоров.

Большинство крупных производителей полупроводниковых устройств на настоящий момент выпускают многоядерные процессоры, в которых несколько вычислительных ядер интегрированы в одной микросхеме. Виртуальные (логические) ядра привносят возможность дополнительного разделения ресурсов, обеспечивая сверхбыстрое переключение контекста между задачами.

Вычислительная мощность многоядерных микропроцессоров зависит от тактовой частоты и числа ядер. Двухъядерные процессоры уже продемонстрировали впечатляющую производительность в ряде устройств (например, в ноутбуках), а сегодня уже доступны процессоры с 16–32 ядрами. Многие из них содержат интегрированные сетевые модули, позволяющие сокращать задержки обработки, традиционно привносимые программным обеспечением сетевых протоколов.

Назначая обработку сетевых пакетов выделенным ядрам, можно добиться на многопортовом оборудовании пропускной способности Gigabit Ethernet, сравнимой с «медью». Это, в свою очередь, способно оказать фундаментальное влияние на стоимость и производительность оконечных устройств, агрегаторов доступа и 3G/4G-оборудования.

Однако преимущества использования многоядерных процессоров в сетевых устройствах не ограничиваются одним увеличением пропускной способности. Использование выделенных ядер для обработки пакетов позволяет избавить от выполнения соответствующих задач основную операционную систему, освобождая вычислительные мощности для выполнения других системных функций. Таким образом, хотя использование многоядерных конфигураций в обработке пакетов в первую очередь находит применение в многопортовой инфраструктурной аппаратуре Gigabit Ethernet, появление недорогих многоядерных процессоров подготавливает почву для разгрузки сетей также на уровне оконечных устройств, устройств доступа и даже клиентского оборудования.

Парадигма многоядерных вычислений

На рис. 1 показана традиционная однопроцессорная система. В такой системе обслуживание всех задач и событий (включая множество неявных функ-

ций обеспечения) возлагается на единственный процессор; при этом задержки переключения контекста могут отрицательно сказываться на общей производительности системы. Эти накладные расходы можно уменьшить применением многопроцессорных систем, где, пока ядро ожидает ввода-вывода, обрабатывает прерывание или занято распределением ресурсов, выполнение может продолжаться на другом ядре.

Симметричная многопроцессорность

Многоядерные системы можно сконфигурировать по-разному. Наиболее типичная конфигурация 2- или 4-ядерной системы – это симметричная многопроцессорность (SMP). *(Здесь симметричная и асимметричная многопроцессорность рассматриваются с точки зрения ОС, а не с точки зрения функциональности самих вычислительных ядер. В симметричной конфигурации на всех ядрах выполняется одна и та же ОС; в асимметричной – различные ОС на различных ядрах. Таким образом, одно и то же ядро может входить одновременно и в симметричную, и в асимметричную конфигурации. – Прим. перев.)* В SMP-системе все ядра рассматриваются как равноправные ресурсы, способные выполнять любую работу. *(Это относится только к «классическому» SMP. В альтернативной SMP-конфигурации используется привязка (affinity) ряда задач к конкретным ядрам, в то время как остальные ядра расцениваются как универсальный пул ресурсов. Такое поведение больше похоже на AMP, но формально причисляется к SMP, т.к. на всех ядрах выполняется одна и та же ОС. – Прим. перев.)* На рис. 2 показана 2-ядерная SMP-система. В ней любая задача может выполняться на любом из ядер, а распределение нагрузки и планирование возложены на ОС.

Преимущества использования SMP могут показаться очевидными: если распределить нагрузку между несколькими ресурсами, то время, требуемое для выполнения задачи, уменьшается. (Есть даже такая поговорка: «много рук – работа легче»). Однако прирост производительности, привносимый простым умножением числа исполнителей, не обязательно будет линейным. К примеру, 2-ядерная система не будет вдвое эффективнее одноядерной, и 4-ядерная тоже не даст 4-кратного выигрыша. *(Это теоретически верно в масштабах всей системы. Однако для конкретных характеристик типа пропускной способности можно добиться линейного масштабирования на конечном участке. – Прим. перев.)* Это происходит потому, что некоторые задачи *обязаны* выполняться последовательно и не могут быть выполнены параллельно. Цитируя «Мифический человеко-месяц» Фредерика Брукса *(Брукс, Ф. «Мифический человеко-месяц, или Как создаются программные системы», Символ-Плюс, 2007 – Прим. перев.)*, «девять женщин не могут выносить ребенка за один месяц». Всегда существует определённый процент задач, подпадающих под это определение, вследствие чего линейное масштабирование производительности в многоядерных системах становится невозможным.

Чтобы выполнять в многопроцессорных вычислительных средах код, изначально написанный для сред однопроцессорных,– а таково подавляющее большинство программных продуктов на сегодняшний день – его нужно дорабатывать. Бóльшая часть современного кода написана с допущением, что доступ к данным будет иметь только один процессор, поэтому никаких мер по предотвращению «гонок» *(в русскоязычной терминологии тж. «критические состязания» – Прим. перев.)* между несколькими процессорами в нём не принято. Конечно, критические секции кода можно выделить и защитить блокировками, и это сделает выполнение его в многопроцессорной среде возможным – но далеко не обязательно оптимальным. Бóльшая часть кода, совместимого с многопроцессорными конфигурациями, в реальности на многопроцессорной системе выполняется медленнее, поскольку требует дополнительных затрат на блокировку и синхронизацию. Наибольшее

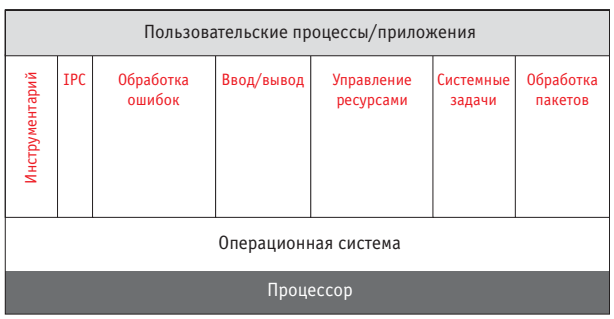

Рис. 1. Традиционная однопроцессорная система

Рис. 2. Симметричная многопроцессорность с двумя процессорами

увеличение производительности на многоядерных процессорах может дать только код, изначально разработанный для параллельного выполнения. Таким образом, выигрыш от применения SMP напрямую зависит от того, насколько ваше ПО «параллелизовано».

Асимметричная многопроцессорность

Существует ещё одна модель многопроцессорности – т.н. асимметричная (AMP). В случае AMP вычислительные ядра равноправными не считаются, и выполнение некоторых задач привязывается к конкретным ядрам. Это даёт сразу несколько выигрышей – в частности, позволяет изолировать обработку прерываний, ввод-вывод и доступ к памяти, а также минимизировать количество переключений контекста для многократно повторяющихся операций. Параллелизм в AMP-системе не менее важен, чем при использовании SMP, но AMP позволяет избавить ряд ядер от накладных расходов по выполнению рутинных функций ОС.

Хорошие кандидаты для выполнения на выделенных ядрах – задачи, которые могут выполняться независимо (или хотя бы преимущественно независимо). Обработка сетевых пакетов – как раз одна из таких задач. Рис. 3 иллюстрирует, как обработку пакетов можно обособить от всех остальных системных процессов и выполнять на выделенном процессоре. Если обособляемое ПО (в нашем случае ПО обработки пакетов) узкоспециализированное, то ему для выполнения не требуется полнофункциональная ОС. Чтобы обеспечивать выполнение специализированного программного модуля, изображенного на рис. 3, достаточно минимального набора сервисов – управление оборудованием, распределение памяти, ввод-вывод, таймеры. В результате выделенные ядра тратят меньше времени на выполнение системных задач и переключение контекста, и больше – на обработку пакетов.

Разделение плоскостей данных и управления

Системы часто проектируются «сверху вниз» – иными словами, сначала функции разделяются на относящиеся к плоскости данных и плоскости управления, а потом уже для каждой из этих функциональных областей прорабатывается соответствующая реализация. К примеру, на рис. 4 изображена система, у которой процессорам 0–3 назначены плоскости управления и они выполняют ОС Linux или ОС реального времени (например, VxWorks) в режиме SMP. Остальные процессоры (4–n) работают в AMP-режиме и выполняют специализированные программные модули, реализующие функции плоскости данных.

Чтобы код обработки пакетов мог работать эффективно, он должен быть написан изначально с учётом специфики многоядерных вычислительных сред. Из этого непосредственно следует, что традиционные стеки TCP/IP здесь не справятся, поскольку в них изначально не предусмотрен необходимый уровень параллелизации алгоритмов.

Системы, выигрывающие от многоядерной обработки пакетов

Многоядерная реализация обработки пакетов – подход кардинально новый. Традиционные стеки TCP/IP не были изначально рассчитаны на использование в многоядерных вычислительных средах (хотя некоторые из них были впоследствии доработаны для повышения эффективности работы на SMP-архитектурах). Пропускная способность системы, использующей многоядерную обработку пакетов, может многократно превышать таковую для однопроцессорной или SMP-системы. Но какие именно системы могут получить от этого выигрыш? Если рассмотреть типы различных задач, выполняемые системой, то можно изобразить гипотетическую процентную шкалу, у которой левому пределу будет соответствовать 0% обработки пакетов, а правому – 100% (рис. 5).

Рис. 3. Использование асимметричной многопроцессорности для обработки пакетов

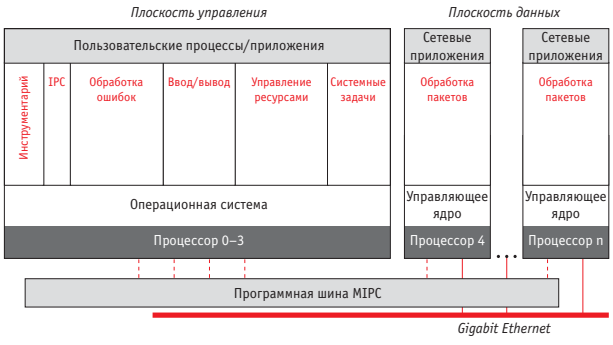

Рис. 4. Вариант реализации плоскости данных и плоскости управления при асимметричной многопроцессорности

Системы, у которых обработка пакетов является основной задачей (включая инфраструктурное оборудование беспроводных и оптических сетей, а также Carrier Ethernet), получат от асимметричного многоядерного подхода наибольший выигрыш. Также выиграют серверы, потому что потребности подписчиков в пропускной способности постоянно растут. По мере того как 3G/4G-приложения становятся всё более «прожорливыми» по части трафика, используя пакетную передачу данных и голоса, в конечном итоге все мобильные устройства оказываются зависимыми от производительности обработки пакетов.

Сегодня первыми апологетами применения многоядерной асимметричной обработки пакетов стали производители сетевого и телекоммуникационного оборудования. Однако вскоре ожидается распространение этой парадигмы и в другие области – в частности в серверные и прочие решения, связанные с интенсивной обработкой пакетов.

Как показано на рис. 6, чтобы разрабатывать эффективные AMP-системы, требуется фундаментальная компетенция сразу в нескольких областях.

Рис. 6. Составляющие многоядерной системы обработки пакетов

Многоядерные процессоры

Современные многоядерные процессоры – это уже не просто несколько процессоров на одном кристалле; в них интегрировано множество дополнительных функций, которые могут (при правильном применении) значительно повысить эффективность SMP/AMP-конфигураций. Аппаратное хеширование, кеширование, межъядерные коммуникации, управление прерываниями и памятью – все эти функции могут работать гораздо быстрее, чем аналогичные программные механизмы, но только если системное ПО умеет их использовать.

Как насчёт логических ядер?

В дополнение к нескольким вычислительным ядрам ряд производителей кремния реализует в рамках каждого физического ядра несколько логических ядер (threads). Такие процессоры хранят и аппаратно поддерживают несколько контекстов одновременно, что позволяет очень быстро переключать их в случае блокировки одного из логических ядер. Это помогает оптимизировать производительность ядер, на которых выполняется несколько процессов, т.к. переключение контекста между процессами занимает много времени – при условии, конечно, что применяемая ОС поддерживает логические ядра.

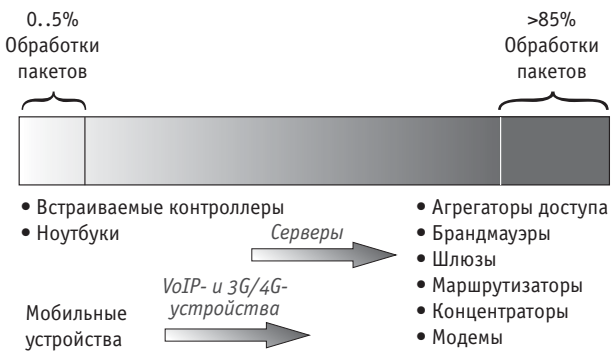

Рис. 5. Распределение вычислительных мощностей

Применение логических ядер позволяет увеличить производительность плоскости управления, где за ресурс процессора конкурирует несколько процессов одновременно. В реализациях плоскости данных более оптимальна обычно модель «выполнение до завершения» (run-to-completion). *(Иными словами, код, реализующий плоскость данных, реже вызывает сброс конвейера. – Прим. перев.)* Таким образом, степень оптимизации производительности плоскости данных будет зависеть от того, сколько вычислений возложено на каждое логическое ядро.

Чтобы эффективно использовать логические ядра, необходимо назначать их процессам так, чтобы они не блокировались по одному и тому же условию. Если одно логическое ядро будет блокировано, второе должно быть готово к выполнению. В 1965 году Эдсгер Дейкстра озвучил проблему пяти компьютеров, конкурирующих за пять разделяемых накопителей на магнитной ленте. Впоследствии этот сценарий многократно перефразировался; одна из, пожалуй, наиболее известных его интерпретаций – это задача пяти обедающих философов, сидящих за круглым столом, когда между каждыми двумя соседями лежит по вилке, и каждому, чтобы есть спагетти из тарелки в центре стола, вилок нужно две. Несмотря на то, что эта ставшая классической

аналогия чаще всего используется для иллюстрации проблемы взаимных блокировок при многопоточном программировании, она также демонстрирует необходимость синхронизации нескольких процессов при организации им доступа к разделяемому ресурсу. В задачах обработки пакетов с использованием логических ядер число таких синхронизаций необходимо минимизировать.

Ещё один важный аспект – это кеширование. Интерпретация логических ядер как отдельных процессоров не учитывает последствий совместного использования кэша. Если ядра, отвечающие за обработку пакетов, работают с различными областями памяти, каждое переключение контекста будет вызывать сброс кэша, что, в свою очередь, отрицательно скажется на производительности.

Операционная система

Операционная система (ОС) играет в многоядерных вычислительных средах ключевую роль. Она должна обеспечивать начальную загрузку для всех ядер, межъядерное взаимодействие, динамическое обновление кода, управление энергопотреблением и переключение контекста. В AMP-системах для координации нескольких ОС важно также иметь эффективный механизм обмена сообщениями.

Не следует забывать и об отладочных средствах. Оптимизация производительности требует наличия диагностического инструментария, способного отслеживать несколько ядер одновременно, централизованно выводить отладочные сообщения и расставлять точки останова на требуемых ядрах. Отладка многоядерной системы может оказаться очень трудной задачей, и если взяться за неё без соответствующего инструментария, вам, возможно, захочется сменить профессию.

Сетевые технологии

То, что для построения эффективной системы обработки пакетов нужно обладать фундаментальными знаниями в области сетевых технологий, вроде бы само собой разумеется. Однако многоядерные сетевые системы содержат дополнительный уровень сложности, который поначалу не настолько очевиден. «Параллелизация» стека протоколов, изначально написанного для однопроцессорной среды, конечно, даст свой результат, но блестящим он не будет. Эффективного управления разделяемыми структурами данных и соединениями и синхронизации плоскостей данных и управления можно добиться только применением стека, разработанного специально для многоядерных сред. В свою очередь, чтобы создать такой сетевой стек в соответствии с современными отраслевыми стандартами, необходимо глубинное понимание протоколов уровней 2, 3, 4 и выше.

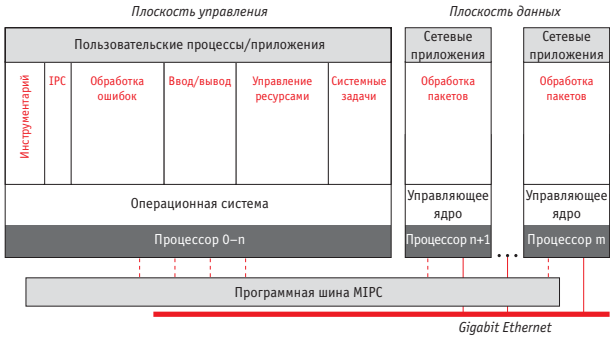

Рис. 7. Асимметричная обработка пакетов: реализация от *Wind River*

Подход к оптимизации ПО обработки пакетов

Компания *Wind River,* долгое время занимающая позицию лидера в области высокопроизводительного встраиваемого ПО, является также пионером оптимизации ПО многоядерных вычислений. Применение подхода *Wind River* к ПО для многоядерных сред в приложениях обработки сетевых пакетов позволяет многократно увеличить их производительность. Однако процесс перехода от однопроцессорных систем к многоядерным весьма нетривиален.

В этой части приводятся типовые проблемы, возможности и компромиссы, с которыми приходится иметь дело при разработке ПО для многоядерных сред.

Функции «короткого пути» (fast path)

Вся оптимизация обработки пакетов за счёт применения нескольких ядер происходит в плоскости данных. Когда соединение устанавливается, оно сначала проходит через плоскость управления, чтобы задать настройки коммутации/маршрутизации и прочие параметры, которые потребуются для последующей передачи пакетов. Смысл в том, чтобы впоследствии переложить как можно больше рутинных повторяющихся операций (анализ/модификация заголовков, шифрование/дешифрование, трансляция адресов, работа с тегами) на плоскость данных, без вмешательства плоскости управления вообще. На рис. 7 приведён пример реализации плоскостей данных и управления при помощи ПО *Wind River*. Модуль обработки пакетов включает в себя оптимизированный стек протоколов, работающий под управлением компактного и быстродействующего специализированного ядра.

Функции плоскости данных

Чтобы в полной мере использовать все преимущества многоядерного процессора, задачи должны «уметь» выполняться параллельно. Это означает, что вычислительные ядра должны быть способны одновременно обрабатывать пакеты, принадлежащие различным потокам данных. Для протоколов без установления соединения (скажем, IP) это можно реализовать использованием разделяемой базы пересылочной информации (Forwarding Information Base – FIB). Для протоколов на основе соединений (скажем, TCP или SCTP) информацию о состоянии необходимо поддерживать отдельно для каждого соединения. *(В частности, разделяемыми должны быть TCB и сокет. Эффективную защиту разделяемых структур данных от «гонок» может обеспечить либо ждущая блокировка (spinlock) на уровне ОС, либо применение атомарных операций на аппаратном уровне. – Прим. перев.)* Если все пакеты из одного и того же потока данных обрабатываются одним и тем же ядром, то они не только будут обработаны в исходном порядке, но и не станут причиной «гонок» между ядрами в процессе обновления информации о состоянии данного соединения. Из этого следует важное понятие в части распределения вычислительных ресурсов – если все пакеты потока данных обрабатываются одним и тем же ядром, говорят, что поток за этим ядром «закреплён».

«Закрепление» потоков данных

Обычно для данных, доступных по записи более чем одному процессору, должен быть предусмотрен механизм блокировки (чтобы избежать «гонок»). Однако применение блокировок в функциях «короткого пути» может снизить производительность на 20% и более, т.к. на время активности блокировок вся параллелизация вычислений фактически сводится на нет.

Чтобы избавиться от этой проблемы, в ряде решений обработки пакетов реализован вариант «закрепления» потоков данных (flow pinning), в котором аппаратно гарантируется, что конкретный поток (идентифицируемый по адресу получателя, 5-tuple *(IP-адрес и порт отправителя, IP-адрес и порт получателя и используемый протокол. – Прим. перев.)* или другому предопределённому критерию) всегда будет перенаправлен для обработки одному и тому же ядру. Это устраняет необходимость в разделении базы пересылочной информации между несколькими ядрами, поскольку каждому ядру достаточно «знать» только о своих собственных соединениях.

«Закрепление» потоков снижает потребность в блокировках в коде, но порождает проблему с балансировкой нагрузки. Если объёмы трафика в потоках переменны, или потоки не являются непрерывными по времени, нагрузка одного ядра постепенно может непропорционально возрасти, в то время как остальные ядра будут простаивать (см. рис. 8). В худшем случае это может привести к снижению общей производительности системы до уровня одного ядра, сводя на нет все преимущества многоядерности.

Чтобы предотвратить или минимизировать разбалансировку нагрузки, можно, например, ввести в систему «диспетчер», измеряющий относительную загрузку ядер и перенаправляющий входные потоки соответственно. Однако потоки далеко не всегда однородны по наполнению и сроку жизни; в результате однозначную метрику для предсказания загрузки системы подобрать очень трудно. Таким образом, если разбалансировка системы превышает критический предел, часто остаётся единственный выход – перезагрузка.

Совместное использование потоков данных

Альтернативой «закреплению» является совместное использование потоков данных (flow sharing). В этой модели любое ядро может обрабатывать пакеты, принадлежащие любому потоку. Это позволяет избежать разбалансировки, поскольку пакету не приходится ждать доступности конкретного ядра – он обрабатывается первым освободившимся. Однако совместное использование потоков подразумевает также и совместное использование пересылочной информации всеми ядрами, имеющими доступ к конкретному соединению или потоку. Проще всего это реализуется для протоколов без установления соединения (например, IP и UDP). Поскольку база пересылочной информации (FIB) и контекст безопасности (security association information) меняются редко, их защита атомарными операциями блокировки на производительность практически не влияет.

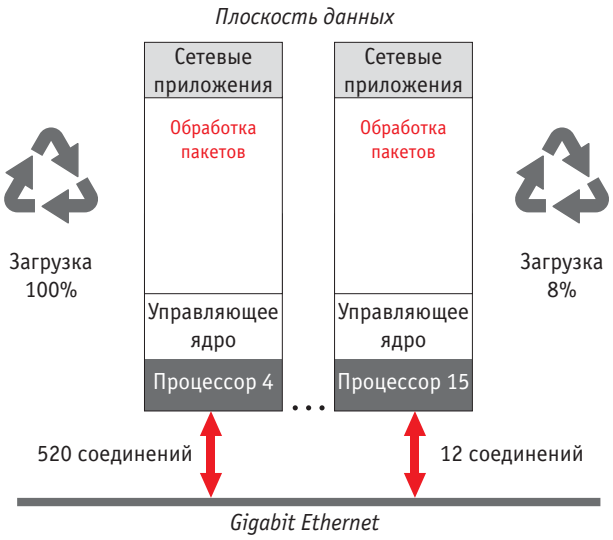

Рис. 8. Разбалансировка нагрузки при «закреплении» потоков данных

Для протоколов на основе соединений (включая TCP и SCTP) информацию о состоянии необходимо обновлять после обработки каждого пакета. В этом случае применение блокировок совершенно не оправдано, т.к. это останавливало бы всю обработку миллионы раз в секунду.

В некоторых многоядерных процессорах реализованы аппаратные механизмы, гарантирующие, что пакеты одного и того же потока будут одновременно обрабатываться не более чем одним ядром. Это даёт возможность эффективно реализовывать атомарные операции, избегая программных блокировок. Ряд процессоров позволяет перенаправить данные со всех входящих портов в общую очередь, доступную для обработки всем ядрам. Это отлично сочетается с концепцией совместного использования потоков, т.к. в данном случае ядра просто «стоят в очереди» за пакетами – как такси в аэропорту (рис. 9).

Такой подход, правда, поддерживается не любыми многоядерными процессорами. Некоторые из них помещают данные по каждому порту в отдельную очередь, которая затем должна опрашиваться модулем обработки пакетов. Если портов много, то опрос всех очередей всеми модулями обработки пакетов может оказаться непрактичным; в этом случае можно назначать группы ядер соответствующим группам портов. При этом риск разбалансировки нагрузки сохраняется, но уменьшается благодаря доступности нескольких ядер одновременно.

Оптимизировать можно только функции «короткого пути». Это вызывает неизбежный вопрос: почему тогда не включить всю сетевую функциональность в «короткий путь»? Ответ прост – от этого «короткий путь» перестанет быть коротким. Чем больше функциональности включено в «короткий путь», тем больше кеширование, управление памятью, арбитраж на шинах и межъядерное взаимодействие влияют на производительность. И именно здесь заключён ключевой момент технического решения – список функций важен, но чтобы определить, будет ли конкретное техническое решение системы обработки пакетов удовлетворять поставленным требованиям, одного списка функций мало.

В рамках «короткого пути» можно выполнять множество разных операций. И если очень мало где потребуются все нижеперечисленные операции, то большинству систем будет необходима как минимум одна или более из них:

- коммутация/маршрутизация IPv4/IPv6;
- шифрование/дешифрование IPsec;
- трансляция адресов (NAT);
- коммутация и работа с тегами VLAN;
- список контроля доступа;
- классификация пакетов/фильтрация содержимого;
- функции MPLS;
- туннелирование;
- обработка 4 уровня (UDP, TCP, SCTP).

Часто повторяющиеся и критичные по времени операции, очевидно, следует включить в «короткий путь»; операции же, выполняемые не так часто, допустимо оставить за его пределами, особенно если их включение в «короткий путь» способно отрицательно повлиять на выполнение остальных.

Тесты производительности

Тесты производительности сродни рейтингам автомобилей по расходу топлива: реальный километраж может не совпадать с заявленным. Впрочем, хотя тесты производительности и подвержены влиянию множества факторов (тактовая частота процессора, размер пакета, число соединений, используемые протоколы, число задействованных ядер, объём кэша и т.д.), они способны дать кое-какую важную информацию о поведении системы.

Основное (и, пожалуй, наиболее очевидное) здесь – удостовериться, что тестирование проводилось на том же процессоре и с теми же протоколами, что и у вас. Если вам необходим IPv6 и IPSec с шифрованием AES 256, не ожидайте адекватных оценок от тестов производительности маршрутизации IPv4. Маленький кэш подразумевает

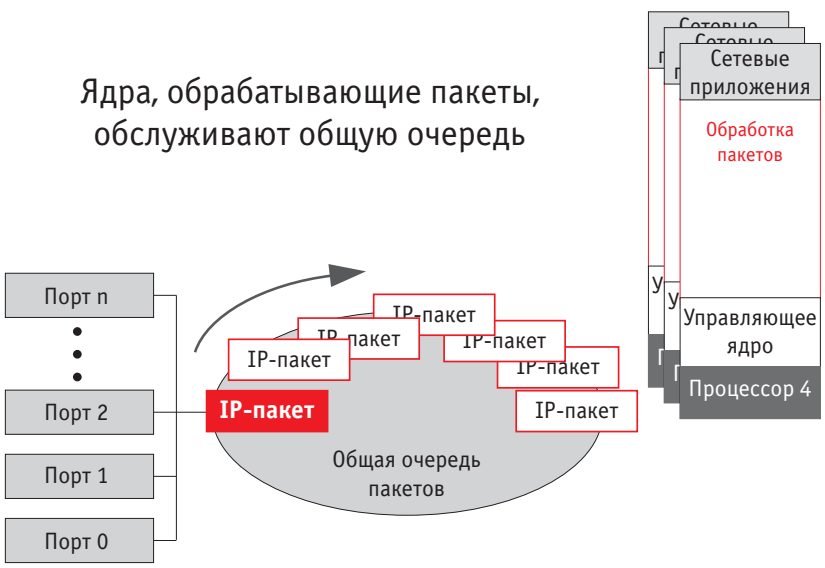

Рис. 9. Совместное использование потоков данных с общей входной очередью

больше операций считывания строк – а значит, и меньшую пропускную способность, чем у систем с большим объёмом кэша. Тактовая частота процессора также может влиять на производительность – если, конечно, система не связана каким-либо другим ограничением (например, по числу портов или объёму памяти).

Также важным (и тем не менее часто упускаемым из виду) фактором является ОС плоскости управления, используемая при тестировании. «Затопление» (flooding) многоядерной системы пакетами может дать хорошее представление о возможностях процессора, но не учесть производительности взаимодействия ОС (скажем, Linux) с кэшем. Чтобы избежать узких мест и потерь производительности в системах с несколькими ОС, необходим низкоуровневый опыт адаптации ОС – в частности, планировщика, подсистемы управления памятью и механизмов межзадачного/межпроцессорного взаимодействия.

Для случая обработки незашифрованных (clear text) пакетов тестирование обычно проводится на пакетах минимального размера, поскольку именно с ними связан максимум накладных расходов. Для Ethernet этот размер составляет 64 байта. После добавления к 64-байтовому кадру 20-байтового заголовка и межпакетных интервалов (согласно спецификации), максимальная пропускная способность канала должна равняться 1 Гбит/с. Это соответствует 1,44 млн пакетов в секунду. Использование пакетов большего размера (128, 256, 512 байт) позволяет использовать меньше пакетов для передачи того же числа бит.

Для зашифрованных пакетов (например, как в случае с IPSec) пропускная способность на пакетах большего размера может быть ниже, чем на мелких – благодаря применяемым механизмам шифрования/дешифрования. Очевидный вывод здесь – «стандартные тесты производительности» совершенно не обязательно покажут правильный результат для вашего конкретного случая, и нужно обязательно учитывать характер трафика и предъявляемые требования. Лучшими индикаторами послужат реальные тесты, проведённые в ваших реальных условиях.

Также следует иметь в виду такой фактор, как масштабируемость решения. Будет ли она линейной, или кривая производительности будет «заваливаться» при превышении определённого числа используемых ядер? В последнем случае это будет означать, что проблема дизайна кроется не в возможностях процессора, и будущие поколения процессоров не помогут её исправить – система «упрётся в потолок» изначально, ещё при рождении. Масштабируемый же дизайн, напротив, позволит получить от неизбежного совершенствования многоядерных процессоров в будущем дополнительный прирост производительности.

Поддерживаемые процессоры

Многоядерные процессоры на сегодняшний день доступны от целого ряда производителей, и число этих производителей продолжает расти. Использование адекватной системы критериев поможет разработчикам подобрать наиболее подходящий процессор для решения поставленных задач.

Однако процессор, подходящий для одного проекта, совершенно не обязательно подойдёт для другого – иногда разные процессоры применяются даже в разных поколениях одного и того же устройства. Стоимость, функциональность, уровень технической поддержки, скорость выхода на рынок, партнёрская экосистема – всё это может измениться за время, сопоставимое с временем жизни поколения устройств. И хотя в любой оптимизированной системе часть кода обязательно будет жёстко привязана к используемому процессору, переносимость ПО многоядерной обработки пакетов даёт надёжную защиту от этих изменений.

Полезная нагрузка кадра Ethernet, байт	Количество байт в кадре (размер кадра + 20)	Кадров в секунду (1 Гбит/с / (Количество байт в кадре × 8 бит в байте))
64	84	1,488,095
128	148	844,595
256	276	452,899
512	532	234,962
1,024	1,044	119,732
1,280	1,300	96,154
1,518	1,538	81,274

Рис. 10. Теоретические максимумы пропускной способности Gigabit Ethernet

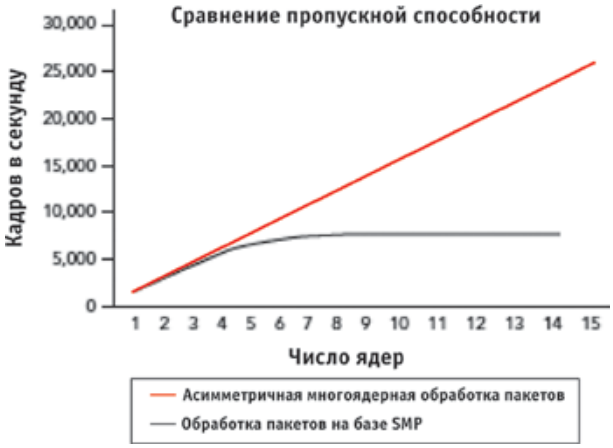

Рис. 11. Результаты тестов производительности

ПО многоядерной обработки пакетов может быть критическим элементом вашей системы. Чтобы максимально продлить жизненный цикл своих инвестиций, будьте уверены, что выбираете решение от партнёра, опыт которого подтверждён многолетним сотрудничеством со множеством производителей полупроводниковых устройств.

Заключение

Многоядерные вычисления относятся к т.н. прорывным технологиям. Многоядерные системы способны обрабатывать пакеты значительно эффективнее одноядерных – но только при условии, что ими управляет оптимизированное ПО. Существующие стеки протоколов не способны обеспечить требуемую производительность в многоядерных средах – чтобы эффективно использовать потенциал многоядерных процессоров в сетевых приложениях, нужна кардинально новая парадигма. Для её реализации, в свою очередь, требуется большой опыт в области процессоров, ОС и сетевых технологий – и именно во всех трёх одновременно. Двух будет недостаточно.

На текущий момент концепция многоядерной обработки пакетов применяется в оконечных устройствах, агрегаторах доступа и базовых устройствах – там, где требуется наибольшая пропускная способность. Однако гибкость и универсальность этого подхода обещает вывести на новые уровни производительности и многие другие классы устройств – вплоть до пользовательских.

Компания *Wind River* находится на переднем крае изменений, вносимых многоядерными технологиями в современную индустрию, и предлагает весь свой богатый опыт в области ПО многоядерных вычислений и средств разработки, чтобы обеспечить вам преимущество в этой новой конкурентной среде. ПО многоядерных вычислений от *Wind River* включает в себя ОС с поддержкой SMP и AMP, встраиваемый гипервизор, управляющий модуль для вычислительных ядер плоскости данных, модули скоростной обработки пакетов и множество других компонентов, а также исчерпывающий набор инструментария для всего цикла разработки многоядерных вычислительных систем. Кроме того, ПО *Wind River* тесно интегрировано и оптимизировано для ведущих современных многоядерных процессоров. Как результат, решения от *Wind River* предоставляют максимум гибкости и позволяют разрабатывать масштабируемые технические решения, удовлетворяющие требованиям как ваших текущих, так и будущих проектов.