Для цитирования: Недбайло Ю. А. Проблемы масштабирования производительности подсистемы памяти многоядерного микропроцессора и методы их решения // Вопросы радиоэлектроники. 2018. № 2. С. 23–31.
УДК 004.318

Ю. А. Недбайло[1, 2]

[1] АО «МЦСТ», [2] ПАО «ИНЭУМ им. И. С. Брука»

ПРОБЛЕМЫ МАСШТАБИРОВАНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ПОДСИСТЕМЫ ПАМЯТИ МНОГОЯДЕРНОГО МИКРОПРОЦЕССОРА И МЕТОДЫ ИХ РЕШЕНИЯ

Пока закон Мура позволяет регулярно наращивать количество ядер, многоядерные микропроцессоры целесообразно делать с распределенным общим кэшем. Их разработка в основном сводится к проектированию подсистемы памяти. С увеличением количества ядер поддержание производительности подсистемы памяти (пропускной способности, времени доступа, качества обслуживания) таких процессоров на необходимом уровне сопряжено с рядом проблем. В статье рассмотрены основные из них, такие как оптимизация времени доступа в распределенный общий кэш, его ассоциативность и разбиение, поддержка когерентности (кодирование и обновление справочника) и архитектура сети соединений на кристалле. Для всех рассмотренных проблем приведены некоторые существующие методы их решения. Анализ и эксперименты позволяют оценить предел эффективной масштабируемости таких процессоров при решении данных проблем рассмотренными методами порядком тысячи ядер.

Ключевые слова: *архитектура, многоядерность, подсистема памяти, общий кэш, когерентность, сеть на кристалле.*

Введение

Самым привлекательным подходом к разработке микропроцессоров, пока позволяет закон Мура, является наращивание количества ядер, что дает наибольший прирост пиковой производительности и сводится к проектированию подсистемы памяти. Однако при этом возникает и все обостряется ряд проблем. Во-первых, память развивается не так быстро – уменьшается ее пропускная способность в пересчете на одно ядро. Во-вторых, усложняется само проектирование – моделирование, физическое проектирование, верификация. Кроме того, чтобы процессор хорошо подходил для самого широкого спектра задач, от него требуется соответствие общепринятым парадигмам, таким как когерентная общая память и однородный доступ к памяти (UMA).

Распространенным подходом к решению этих проблем является создание процессоров с распределенным общим кэшем, соединенных сетью на кристалле (рис. 1). Они имеют достаточно однородную структуру, что позволяет наращивать число ядер без большого усложнения проектирования и отладки, попутно увеличивая доступный каждому ядру объем кэша, что смягчает проблему медленной памяти. Если к тому же процессор не сделан из отдельных кластеров, он остается близок к UMA и хорошо совместим с параллельными программами.

Сегодня этот подход хорошо отработан на процессорах с 10–20 ядрами. Существуют процессоры с сотнями и даже тысячами ядер, но они скорее специализированные. В данной статье попытаемся ответить на вопрос, насколько возможно дальнейшее масштабирование такого процессора, чтобы он оставался достаточно универсальным, и как этого можно достичь.

Распределенный общий кэш

Каждое ядро обычно имеет свои – приватные – кэши для быстрого доступа к наиболее часто

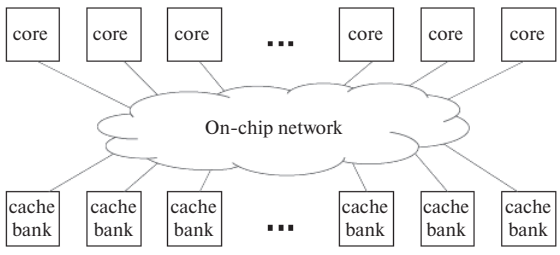


Рисунок 1. Концепция распределенного общего кэша

используемым данным. Однако основной объем кэшей лучше делать общим, динамически распределяемым между всеми ядрами. Чем больше доступный объем кэшей, тем меньше процент промахов в них (рис. 2) и, соответственно, больше фактически доступная ядрам пропускная способность памяти. Чтобы сам кэш не ограничил пропускную способность, его делят на банки, распределенные по кристаллу, тогда узким местом может стать только сеть, соединяющая ядра и банки кэша. Основные проблемы масштабируемости самого кэша – время доступа, ассоциативность и поддержка когерентности.

Оптимизация времени доступа

Время доступа в распределенный кэш увеличивается с количеством банков, поскольку увеличивается число шагов, совершаемых по сети. В традиционной схеме (S-NUCA) положение кэш-строки (home-банк) определяется ее физическим адресом, т.е. случайной динамикой выделения памяти. Чтобы избежать этой случайности, можно выделить три подхода к расположению данных вблизи использующих их ядер:

1. Привязка страниц памяти к банкам кэша силами ОС.
2. Создание копии строки в ближайшем банке при обращении к ней.
3. Миграция строки в ближайший банк при обращении к ней.

При первом подходе физический адрес запроса программы в память формируется из виртуального путем обращения в таблицу страниц. В этой же таблице можно хранить и номер home-банка этой страницы, выбрав ближайший к ядру при ее выделении, а не вычислять его из адреса. Более сложный вариант оптимизации – R-NUCA [1] – назначает странице несколько банков с интерливингом между ними (Rotational Interleaving) (рис. 3). Такой подход имеет два недостатка. Во-первых, он требует модификации ОС и какой-то эвристики, определяющей, какие страницы нужно хранить вблизи ядра, а какие – рассеивать по банкам. Во-вторых, оптимизация перестанет хорошо работать при случайной пересадке программ на другие ядра, что часто бывает в многозадачной ОС.

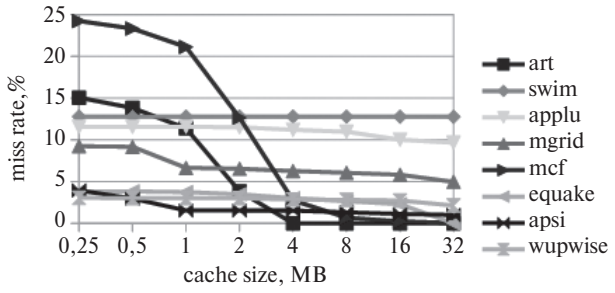


Рисунок 2. Частота промахов в кэш в зависимости от его объема в некоторых тестах пакета SPEC CPU2000 на симуляторе процессора «Эльбрус»

При втором подходе, когда ядро запрашивает данные, их копия создается не только в home-банке, определяемом адресом, но и в ближайшем к ядру. Обращение в кэш начинается с ближайшего к ядру банка и затем, при промахе, отправляется в home-банк. Необходимость хранить копию в home-банке обусловлена инклюзивностью общего кэша, нужной для поддержки когерентности, но создание дополнительных копий фактически уменьшает объем кэша. По этой причине такой подход мало полезен и требует подсказок или эвристики, определяющих, какие данные нуждаются в дублировании; некоторый механизм этого предложен в [2].

Третий подход избавлен от недостатков предыдущего путем отказа от инклюзивности и использования отдельного механизма когерентности. Стоимость такого механизма окупается повышением эффективности кэша [3], поэтому третий подход выглядит привлекательным. Среди его вариаций стоит упомянуть Victim-миграцию [4], кооперативное кэширование [5] и CloudCache [6]. Они подразумевают заведение строки в ближайшем банке при обращении к ней и затем перемещение ее в более дальние банки. Каждый банк делится на приватную и общую части (рис. 4), последние выполняют роль victim-кэша для приватных частей.

Более подробно эти оптимизации рассмотрены в [7]. Теоретически они должны поддерживать время доступа к приватным данным постоянным, и только общие данные (обычно редко используемые) будут испытывать его увеличение в соответствии с ростом задержек сети. В экспериментах на 16-ядерной модели процессора «Эльбрус» нами был оценен эффект, как от роста задержек, так и от оптимизаций, в несколько процентов, что обещает беспроблемное увеличение количества ядер еще как минимум на порядок.

Ассоциативность и разбиение

Ассоциативностью кэша обычно называют количество позиций, в которых могут храниться данные определенного адреса, т.е. размер сета при сет-ассоциативности. Чем ассоциативность больше, тем менее вероятно, что данные нескольких одновременно обрабатываемых массивов будут вытеснять друг друга, что может быть проблемой при большом количестве ядер. В особенности это важно, когда используется разбиение – выделение определенной доли объема кэша определенным ядрам или типам обращений – для защиты процессов реального времени от влияния остальных или для повышения эффективности кэша.

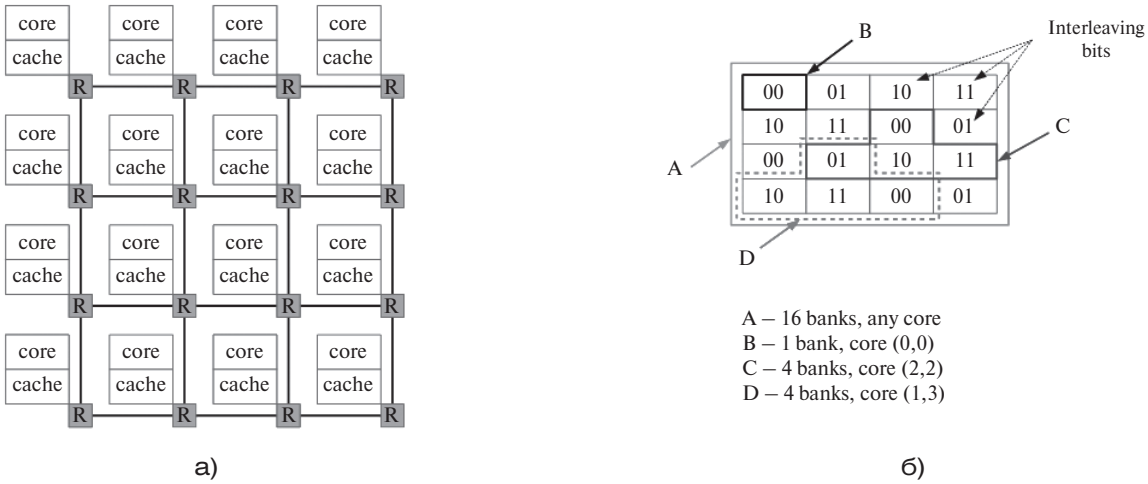


Рисунок 3. R-NUCA: а – «плиточная» топология; б – Rotational Interleaving

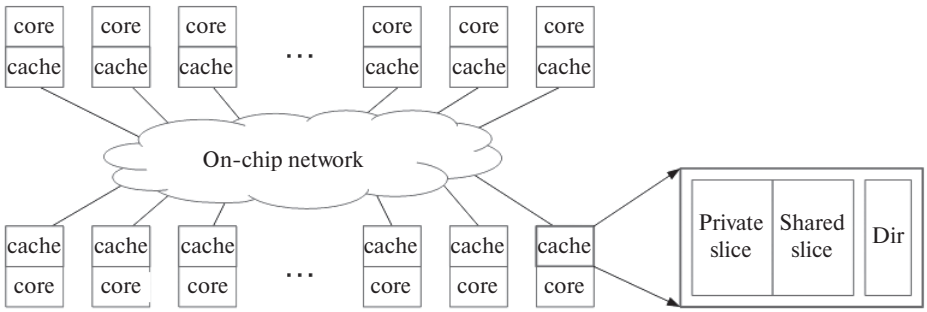


Рисунок 4. Деление общего кэша на приватную и общую части и справочник

Традиционный способ разбиения – столбцами (way-partitioning, WP). Например, сет-ассоциативный кэш с 32 строками в сете можно разделить между 16 ядрами, назначив каждому ядру два столбца. Такое разбиение является грубым – вместо него можно реализовать механизм Partitioning First, выбирающий в сете позиции, занятые данными разделов, превышающих свою квоту. В любом случае сет-ассоциативный кэш должен иметь много столбцов в соответствии с возможным количеством разделов, т.е. ядер, иначе фактическая ассоциативность будет низкой. Поскольку делать больше нескольких десятков столбцов затратно, это может ограничить масштабируемость кэша порядком сотни ядер.

Радикально решить эти проблемы помогает скошенная ассоциативность – вычисление позиции, соответствующей адресу данных, в каждом столбце кэша своей хэш-функцией. В этом случае сеты теряют привычный смысл: даже если два адреса соответствуют одной позиции в каком-то столбце, в других, скорее всего, они попадают в разные, и обнаруживается, что ассоциативность определяется количеством не столбцов, а кандидатов на замещение. Можно проверить скошенные сеты каждой кэш-строки, занимающей возможную позицию размещения новой, выбрать в них жертву и переместить на ее место кэш-строку из исходного сета (рис. 5). Эта оптимизация называется ZCache [8]; с ней количество кандидатов (и фактическая ассоциативность кэша) может достигать квадрата количества столбцов, а если делать не одно перемещение, а цепочку, – еще больше.

Остается найти ответ на вопрос, как реализовать в таком кэше разбиение. В ZCache можно использовать Partitioning First и различные другие механизмы, но самым масштабируемым, по-видимому, является Futility Scaling (FS) [9]. Он заключается в том, что возраст каждой кэш-строки (при скошенной ассоциативности он глобальный, публикации

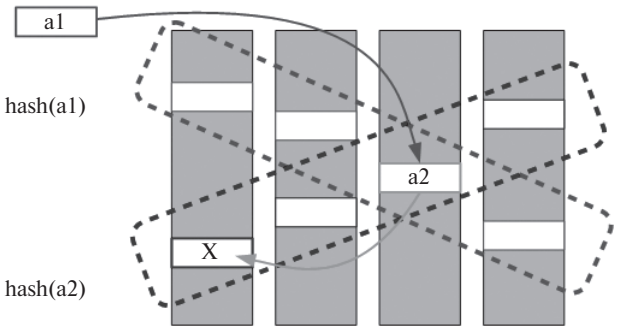


Рисунок 5. Принцип работы ZCache

о ZCache предполагают хранить время последнего обращения) при выборе кандидата умножается на коэффициент, соответствующий ее разделу. Таким образом, разбиение не уменьшает количество кандидатов, и его реализация во всех смыслах хорошо масштабируется.

Мы сравнили традиционный кэш с WP и ZCache с FS на 16-ядерной модели процессора «Эльбрус» (рис. 6), и получилось, что первый требует 32 столбца для оптимальной производительности, а последнему хватает даже двух. Теоретически ZCache и FS масштабируются как минимум до порядка тысячи ядер. В то же время дополнительные просмотры кэша и, возможно, более низкая производительность (на 4% в наших экспериментах) делают ZCache менее предпочтительным при небольшом количестве ядер.

Поддержка когерентности

Обычно подразумевается, что каждый кэш системы хранит только актуальные копии – это называется когерентностью. Масштабируемой ее реализацией является использование распределенного справочника, в роли которого обычно выступает распределенный общий кэш. Запись в кэше обычно содержит тэг – адрес и состояние кэш-строки – и данные. Если добавить в тэг информацию о наличии копий в других кэшах, такой кэш приобретает и функцию справочника. Логически отделить справочник от кэша (чего требуют некоторые оптимизации времени доступа) проще всего, расширив массив тэгов. Рассмотрим проблемы реализации такого справочника.

Кодирование маски фильтра. Первой проблемой реализации справочника с точки зрения масштабируемости является кодирование информации о копиях. В небольших системах она включает битовый массив – маску фильтра когерентности, каждый бит которого соответствует одному ядру. Уже при порядка сотни ядер хранение такой маски весьма затратно. Первый вариант решения проблемы – загрубление этой маски. Если один бит будет соответствовать нескольким ядрам, потребуется пропорционально меньшее количество бит, но соответственно увеличится и трафик запросов на проверку копий. Когда копий мало, можно хранить их точные номера. Второй вариант – создание дополнительных записей в справочнике, когда копий много. Тогда, например, для 1024 ядер и 64 столбцов тэг должен хранить 16 10-битных номеров – 20 байт информации, что уже приемлемо. А если сочетать оба способа уменьшения маски, объем информации можно сократить еще в несколько раз.

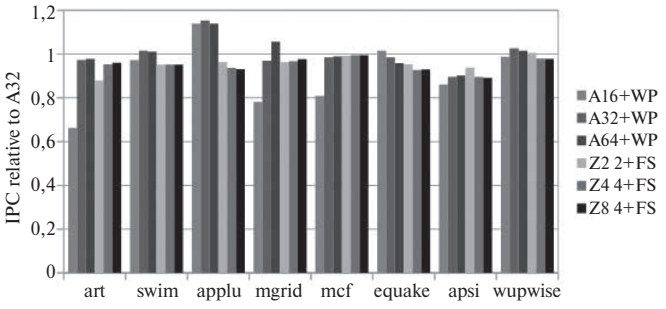


Рисунок 6. Производительность на 16 потоках обычного кэша (с 16–64 столбцами и WP) и ZCache (с 2–8 столбцами и FS) относительно обычного кэша без разбиения

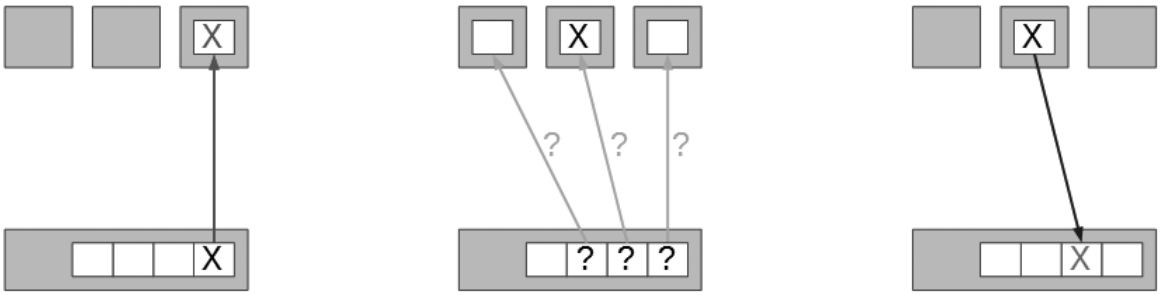
Обновление справочника. Вторая проблема – как поддерживать актуальность справочника? Обычно для этого применяется гашение копий в кэшах при замещении записи в справочнике (back invalidation) (рис. 7а). Поскольку возраст записи в справочнике, вообще говоря, не связан с возрастом копий в кэшах, замещение записи в справочнике часто приводит к гашению активно используемых копий, что снижает эффективность работы кэшей.

Одним из подходов является опрос кэшей справочником – проверка состояния копий для различных записей справочника (рис. 7б). Для гарантированного нахождения неактуальной записи (чтобы не гасить живые копии) с каждым замещением нужно опросить столько ядер, сколько в справочнике кандидатов на замещение, что масштабируется плохо. В качестве компромисса, во-первых, можно проверять только старые записи и, во-вторых, делать это не при замещении, а фоново, по принципу «сборки мусора». Это не устранит проблему полностью, но зато обойдется дешево. Другой подход заключается в том, что ядра оповещают справочник о замещении в кэшах нижнего уровня ядер (рис. 7в). Это примерно удваивает трафик запросов в общий кэш, но зато информация в справочнике точная, и гашение копий происходит редко. Можно сочетать оба подхода, не посылая оповещение, когда трафик запросов слишком высок, и делая «сборку мусора», когда он низок. Это будет и дешево, и все недостатки будут проявляться редко.

Сеть соединений на кристалле

Мы рассмотрели внутреннее устройство банков распределенного кэша, но проблема соединения ядер и банков кэша не менее серьезна. Чтобы не ограничивать производительность в каких-либо задачах, сеть соединений должна обладать рядом качеств:

- достаточной пропускной способностью для обменов каждого ядра с каждым банком;
- минимальными задержками между входом пакета в сеть и выходом, чтобы время доступа в кэш оставалось достаточно низким;



а) б) в)

Рисунок 7. Обновление справочника: а – гашение копий; б – опрос кэшей; в – оповещение

- качеством обслуживания (QoS) абонентов сети (т.е. ядер), которое должно удерживаться в некоторых рамках, чтобы работа одних ядер не вызывала простой других;
- соблюдением порядка пакетов между каждой парой узлов (для тех пакетов, которые этого требуют), иначе некоторые транзакции придется задерживать.

Пропускная способность сети в основном определяется ее топологией – порядком, в котором соединены каналами ее узлы (роутеры), и шириной этих каналов. Самые распространенные топологии, которые хорошо масштабируются, показаны на рис. 8. Соблюдение порядка достигается использованием одного маршрута между каждой парой узлов и FIFO-реализацией всех буферов. Сложными проблемами остаются минимизация задержек и реализация QoS, причем задержки становятся проблемой все более острой [10].

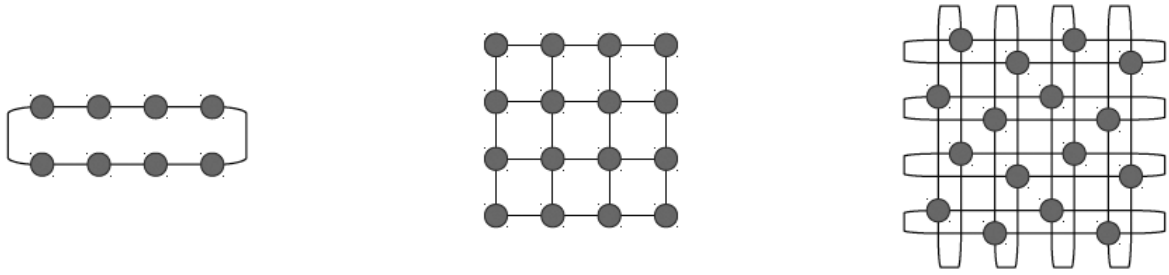
Минимизация задержек

Простейший вариант реализации сети на кристалле, ориентированный на минимальные задержки, – плиточная (tiled) организация – топология, представляющая двумерную сетку с подключением одного ядра и одного банка кэша к каждому роутеру (рис. 3а). Однако если учитывать, что прохождение пакета через каждый роутер включает задержки на его маршрутизацию, арбитраж и прочее, более предпочтительной может оказаться сеть с более крупным шагом и несколькими ядрами и банками на один роутер. В любом случае задержки получаются существенно выше идеальных – определяемых расстоянием.

В борьбе за приближение задержек к идеальным можно выделить два подхода. Первый заключается в оптимизации всех стадий конвейера обработки пакета либо для их ускорения, либо для выполнения параллельно друг с другом. При этом в каждом роутере все равно будет некоторая задержка. Второй подход заключается в реализации экспресс-каналов или их производных. Экспресс-каналы – дополнительные каналы, соединяющие удаленные узлы для уменьшения среднего количества роутеров, посещаемых пакетами, и, соответственно, суммарных задержек в роутерах [11]. Обычные экспресс-каналы подразумевают прокладку на кристалле дополнительных проводов и усложнение структуры роутеров, что дорого. Избавлены от этого экспресс-виртуальные каналы (EVC). Если цепочку обычных каналов соединить байпасами (например, из левого входа роутера в правый выход), включаемыми заранее посылаемым сигналом, эта цепочка работает почти как обычный экспресс-канал (рис. 9). Соединив так попарно все узлы в каждой строке и каждом столбце сети, любой пакет можно передавать в два логических шага, хотя физически сеть остается обычной сеткой, и роутеры усложняются незначительно [12].

Реализация качества обслуживания

Применительно к сетям качество обслуживания может означать следующее:



а) б) в)

Рисунок 8. Топологии: а – кольцо; б – сетка; в – сложенное двумерное кольцо

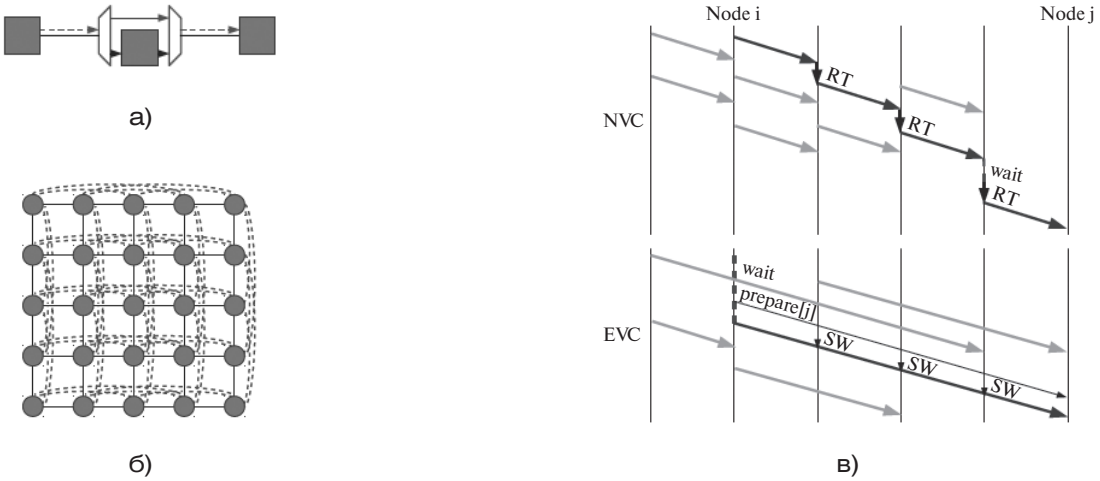


Рисунок 9. Экспресс-виртуальные каналы: а – принцип реализации; б – сеть с каналами; в – пространственно-временнàя диаграмма передачи пакета в сравнении с обычными (NVC) каналами

- Гарантированный сервис – определенную пропускную способность или/и задержки для конкретных абонентов.
- Приоритеты между абонентами при нехватке пропускной способности на всех.
- Честность в распределении пропускной способности при ее нехватке.
- Некоторые более специфические вещи [11].

Первые два качества бывают нужны задачам реального времени и критически важным процессам. Обычным задачам важна честность сети. Программист вправе ожидать, что одинаковые участки кода, запущенные на разных ядрах, выполнятся за примерно одинаковое время, иначе распределение и синхронизация работы между тредами могут быть сопряжены со значительной долей простоев некоторых ядер. Кроме того, когда одновременно выполняются задачи с разной интенсивностью обменов с памятью, суммарная производительность выше, если приоритет отдается менее интенсивным.

Такая честность, или оптимальность распределения пропускной способности, в простых случаях реализуется Round Robin арбитрами. Для сложной сети традиционное решение – Fair Queuing – предполагает перед каждым выходом сети отдельные очереди для каждого потока и арбитр между ними, но этот вариант плохо масштабируется. Другой подход – динамическое определение приоритетов между потоками и учет их арбитрами каждого роутера. Так, схема Preemptive Virtual Clock (PVC) [13] использует счетчики пакетов для каждого потока в каждом роутере. Ее масштабируемость также ограничена из-за площади, которую требуют эти счетчики. Еще одним подходом может быть Source Throttling – торможение тех абонентов, которые используют сеть слишком активно, но в публикациях на эту тему [14] не рассматриваются нечестные сети.

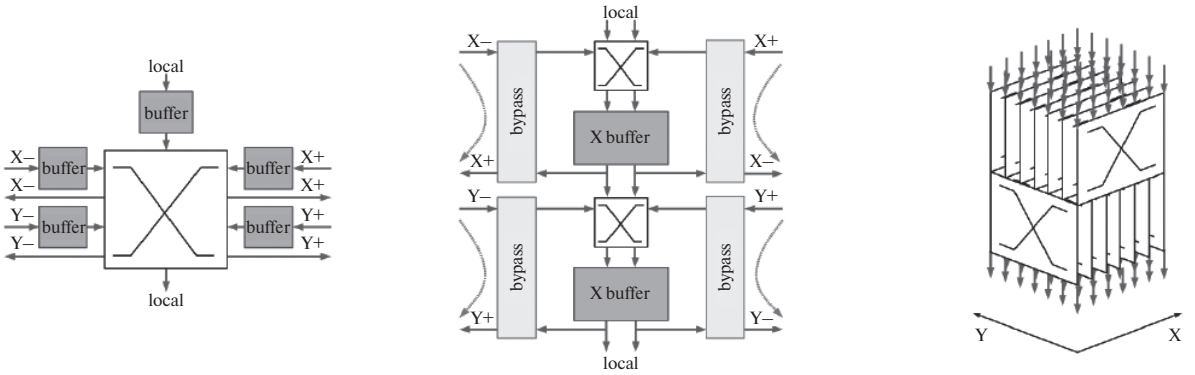
Архитектура сети соединений, сочетающая минимальные задержки, честность и хорошую масштабируемость, предложена в [15]. Опишем ее вкратце.

Архитектура сети FEMIDA

Традиционное устройство роутера подразумевает FIFO-буфер на каждом из пяти входов (от соседних роутеров плюс один от абонентов), коммутатор 5×5 между выходами этих буферов и выходами роутера и управляющую логику (рис. 10а). Но если реализовать экспресс-виртуальные каналы для передачи каждого пакета в каждом измерении сети (сначала *X*, затем *Y*) за один шаг, роутер резонно разделить на две части (рис. 10б), что фактически разделит сеть на два уровня распределенных коммутаторов (рис. 10в).

При таком разделении в каждой части можно реализовать Fair Queuing или любой другой затратный механизм качества обслуживания, и его стоимость будет пропорциональна не общему числу его узлов, а лишь его корню. В FEMIDA примерно половина буфера, общего для половины роутера, статически делится между всеми абонентами строки или столбца, а остальная часть выделяется динамически тем из них, кому их доли не хватает. Таким образом, буфера глубиной, пропорциональной корню из количества узлов, хватает и для реализации честности сети, и для высокой пропускной способности. Порядок пакетов поддерживается LRU-арбитрами, основанными на бинарном дереве сравнений, которое тоже хорошо масштабируется. Для честности сети при высоком трафике реализован механизм предотвращения голодания, аналогичный Source Throttling.

Оценка зависимости площадей простейшего NVC и традиционного EVC роутеров с механизмом



а) б) в)

Рисунок 10. Структура роутеров: а – традиционная; б – предлагаемая; в – иерархия сети

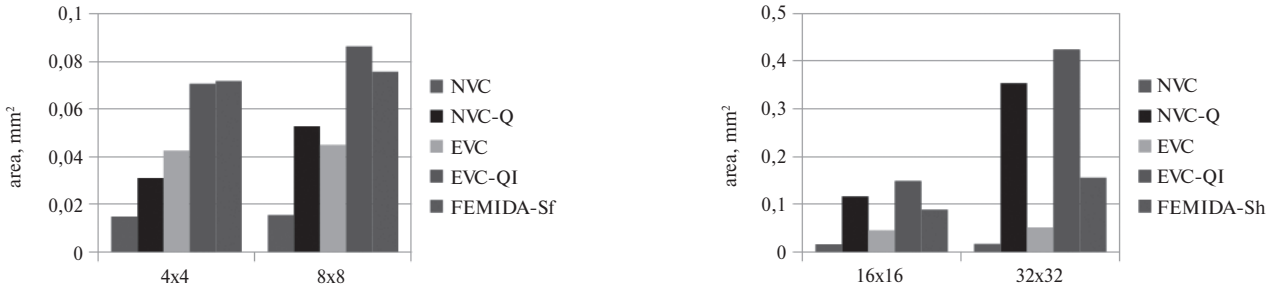


Рисунок 11. Зависимости площадей роутеров NVC, EVC и FEMIDA от размера сети

PVC (–Q) и без него, а также площади оптимальной конфигурации роутера FEMIDA от размера сети приведена на рис. 11. FEMIDA, судя по результатам экспериментов, масштабируется вплоть до 32×32, т.е. 1024 ядер, что достаточно хорошо. Задержки сети (около 30–50 тактов в обе стороны при каналах 1–2 такта) при этом сравнимы с временем доступа в L3-кэш, который обычно и делают общим, и, соответственно, примерно вдвое его увеличат. Без оптимизаций кэша это может снизить производительность на десятки процентов, а с оптимизациями такое падение будет только при активной работе с общими данными, что обычно бывает редко.

Заключение

Многоядерные процессоры лучше всего делать с распределенным общим кэшем. Проанализировав известные проблемы масштабирования основных узлов такой системы, мы видим способы эффективного их решения до порядка тысячи ядер.

Конечно, рассмотренные нами способы не единственно возможные и не всегда самые оптимальные, особенно при небольшом числе ядер, да и список проблем не исчерпывающий – некоторые специфические мы оставили вне поля зрения, некоторые еще, наверно, обнаружат себя, когда процессоры с сотнями ядер станут массовыми.

СПИСОК ЛИТЕРАТУРЫ

1. Hardavellas N., Ferdman M., Falsafi B., Ailamaki A. Reactive NUCA: Near-optimal block placement and replication in distributed caches. SIGARCH Comput. Archit. News, 2009, no. 37 (3), pp. 184–195.
2. Beckmann B.M., Marty M.R., Wood D.A. ASR: Adaptive selective replication for CMP caches. Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '39, Washington, DC, USA, 2006, pp. 443–454.
3. Кожин А.С., Недбайло Ю.А. Оптимизация общего кэша третьего уровня микропроцессора «Эльбрус-8С» // Вопросы радиоэлектроники. 2015. № 3 (3). С. 21–30.
4. Asanovic K., Zhang M. Victim migration: Dynamically adapting between private and shared CMP caches. MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, Massachusetts, USA, 2005, 17 p.
5. Herrero E., Gonzalez J., Canal R. Distributed cooperative caching. Proceedings of the 17th international conference on Parallel architectures and compilation techniques. Toronto, Ontario, Canada, ACM New York, NY, USA, 2008, pp. 134–143.
6. Lee H., Cho S., Childers B.R. CloudCache: Expanding and shrinking private caches. Proceedings of the 2011 IEEE17th International Symposium on High Performance Computer Architecture. HPCA '11, Washington, DC, USA, 2011, pp. 219–230.
7. Кожин А.С., Недбайло Ю.А. Методы оптимизации времени доступа в общий кэш многоядерного процессора // Вопросы радиоэлектроники. 2017. № 3. С. 27–32.

8. Sanchez D., Kozyrakis Ch. The ZCache: Decoupling ways and associativity. Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '43, Washington, DC, USA, 2010, pp. 187–198.
9. Wang R., Chen L. Futility scaling: High-associativity cache partitioning. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47, Washington, DC, USA, 2014, pp. 356–367.
10. De Micheli G., Benini L. Networks on chips: Technology and tools. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2006, 408 p.
11. Dally W., Towles B. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2003, 550 p.
12. Kumar A., Peh Li-S., Kundu P., Jha N. K. Express virtual channels: Towards the ideal interconnection fabric. Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA '07, New York, USA, 2007, pp. 150–161.
13. Grot B., Keckler S. W., Mutlu O. Preemptive Virtual Clock: a Flexible, Efficient, and Costeffective QOS Scheme for Networks-on-Chip. Proceedings 42nd Annual IEEE/ACM Int'l Symp. Microarchitecture (MICRO 42), ACM, 2009, pp. 268–279.
14. Ebrahimi E., Lee C. J., Mutlu O., Patt Y. N. Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multicore Memory Systems. ACM Trans. Comput. Syst., 2012, no. 30 (2), pp. 1–35.
15. Недбайло Ю. А. Разработка сети на кристалле для перспективных многоядерных микропроцессоров // Труды МФТИ. 2017. № 9 (2). С. 151–163.

ИНФОРМАЦИЯ ОБ АВТОРЕ

Недбайло Юрий Александрович, старший инженер, АО «МЦСТ», ПАО «ИНЭУМ им. И. С. Брука», 119334, Москва, ул. Вавилова, д. 24, тел.: 8 (916) 936-86-70, e-mail: nonsens@inbox.ru.

For citation: Nedbailo Yu. A. Memory subsystem performance scaling problems in chip multiprocessors and their solution. Voprosy radioelektroniki, 2018, no. 2, pp. 23–31.

Yu. A. Nedbailo

MEMORY SUBSYSTEM PERFORMANCE SCALING PROBLEMS IN CHIP MULTIPROCESSORS AND THEIR SOLUTION

As long as Moore's law allows to increase the number of cores, it is reasonable to design a chip multiprocessor with a distributed shared cache. The main part of the design process is that of the memory subsystem. As the number of cores increases, maintaining the memory subsystem's performance (throughput, latency, quality of service) at a necessary level in this type of processors is hampered by various problems. The paper discusses the main problems, such as NUCA optimizations in a distributed shared cache, its associativity and partitioning, coherence support (directory encoding and update), and on-chip interconnection network design. For every problem in question, some of the existing methods of their solution are described. Analysis and experiments let us estimate the limit of effective scaling for this type of processors using the methods considered at an order of 1000 cores.

Keywords: architecture, many-core, memory subsystem, shared cache, coherence.

REFERENCES

1. Hardavellas N., Ferdman M., Falsafi B., Ailamaki A. Reactive NUCA: Near-optimal block placement and replication in distributed caches. *SIGARCH Comput. Archit. News,* 2009, no. 37 (3), pp. 184–195.
2. Beckmann B. M., Marty M. R., Wood D. A. ASR: Adaptive selective replication for CMP caches. *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '39,* Washington, DC, USA, 2006, pp. 443–454.
3. Kozhin A. S., Nedbailo Yu. A. Optimizing the inclusive shared L3 cache in Elbrus-8S microprocessor. *Voprosy radioelektroniki,* 2015, no. 3 (3), pp. 21–30 (In Russian).
4. Asanovic K., Zhang M. Victim migration: Dynamically adapting between private and shared CMP caches. *MIT Computer Science and Artificial Intelligence Laboratory,* Cambridge, Massachusetts, USA, 2005, 17 p.
5. Herrero E., Gonzalez J., Canal R. Distributed cooperative caching. *Proceedings of the 17th international conference on Parallel architectures and compilation techniques,* Toronto, Ontario, Canada, ACM New York, NY, USA, 2008, pp. 134–143.
6. Lee H., Cho S., Childers B. R. CloudCache: Expanding and shrinking private caches. *Proceedings of the 2011 IEEE17th International Symposium on High Performance Computer Architecture. HPCA '11,* Washington, DC, USA, 2011, pp. 219–230.
7. Kozhin A. S., Nedbailo Yu. A. Methods of shared cache access latency optimization in chip multiprocessors. *Voprosy radioelektroniki,* 2017, no. 3, pp. 27–32 (In Russian).
8. Sanchez D., Kozyrakis Ch. The ZCache: Decoupling ways and associativity. *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '43,* Washington, DC, USA, 2010, pp. 187–198.
9. Wang R., Chen L. Futility scaling: High-associativity cache partitioning. *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture,* MICRO-47, Washington, DC, USA, 2014, pp. 356–367.
10. De Micheli G., Benini L. Networks on chips: Technology and tools. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. 2006, 408 p.
11. Dally W., Towles B. Principles and Practices of Interconnection Networks. CA, USA, Morgan Kaufmann Publishers Inc. San Francisco, 2003, 550 p.
12. Kumar A., Peh Li-S., Kundu P., Jha N. K. Express virtual channels: Towards the ideal interconnection fabric. *Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA '07,* New York, USA, 2007, pp. 150–161.

13. Grot B., Keckler S. W., Mutlu O. Preemptive Virtual Clock: a Flexible, Efficient, and Costeffective QOS Scheme for Networks-on-Chip. *Proceedings 42nd Annual IEEE/ACM Int'l Symp. Microarchitecture (MICRO 42), ACM,* 2009, pp. 268–279.
14. Ebrahimi E., Lee C. J., Mutlu O., Patt Y. N. Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multicore Memory Systems. *ACM Trans. Comput. Syst.,* 2012, no. 30 (2), pp. 1–35.
15. Nedbailo Yu. A. On-chip network design for prospective chip multiprocessors. *Trudy MFTI,* 2017, no. 9 (2). pp. 151–163 (In Russian).

AUTHOR

Nedbailo Yuriy, senior engineer, JSC MCST, PJSC Brook INEUM, 24, ulitsa Vavilova, Moscow, 119334, Russian Federation, tel.: +7 (916) 936-86-70, e-mail: nonsens@inbox.ru.