**2008**. 35th International Symposium **on Computer Architecture**

2008. ISCA '08. 35th International Symposium on

**ISCA 2008: 35th International Symposium on Computer Architecture, Beijing, China** | June 2008

|  |
| --- |
| ISCA '08 Proceedings of the 35th Annual International Symposium on Computer Architecture |
| Pages 63-74 **Beijing, China**  June 21 - 25, 2008 |

**Пакетное планирование с учетом параллелизма: повышение производительности и справедливости совместно используемых систем DRAM**

Онур Мутлу Томас Москиброда

Microsoft Research

{onur, moscitho} @ microsoft.com

**Аннотация**

В системе с чип-мультипроцессором (CMP) система DRAM распределяется между ядрами. В общей системе DRAM запросы от потока могут не только задерживать запросы от других потоков, вызывая конфликты между банком/шиной/буфером строк, но они также могут разрушить параллелизм уровня DRAM-банка других потоков. Запросы, задержки которых в противном случае были бы перекрыты, могли бы эффективно стать сериализованными. В результате снижается как справедливость, так и пропускная способность системы, и некоторые потоки могут голодать в течение длительных периодов времени.

В этой статье предлагается принципиально новый подход к разработке общего контроллера DRAM, который обеспечивает качество обслуживания потоков, а также повышает пропускную способность системы. Конструкция нашего пакетного планировщика с учетом параллелизма (PAR-BS) основана на двух ключевых идеях. Во-первых, PAR-BS обрабатывает запросы DRAM партиями, чтобы обеспечить справедливость и избежать истощения запросов. Во-вторых, для оптимизации пропускной способности системы PAR-BS использует политику планирования DRAM с учетом параллелизма, которая направлена на параллельную обработку запросов из потока в банках DRAM, тем самым сокращая время задержки, связанное с памятью, испытываемое потоком. PAR-BS плавно включает поддержку приоритетов потоков системного уровня и может предоставлять различные уровни обслуживания, включая чисто условное обслуживание, потокам с различными приоритетами.

Мы оцениваем компромиссные решения, связанные с PAR-BS, и сравниваем его с четырьмя ранее предложенными проектами планировщика DRAM для 4-, 8- и 16-ядерных систем. Наши оценки показывают, что в среднем более 100 4-ядерных рабочих нагрузок PAR-BS улучшает справедливость в 1,11 раз и пропускную способность системы на 8,3% по сравнению с лучшим из предыдущих методов планирования с использованием Stall-Time Fair Memory (STFM). Основываясь на простых правилах приоритезации запросов, PAR-BS также проще реализовать, чем STFM.

1. **Введение**

Система памяти DRAM является основным общим ресурсом для множества процессорных ядер в многопроцессорной системе (CMP). При доступе к этому общему ресурсу разные потоки, работающие на разных ядрах, могут задерживать друг друга, потому что доступ из одного потока может вызвать дополнительные конфликты банка DRAM, конфликты буфера строк и конфликты шины данных / адресов с доступом из другого потока. Кроме того, как мы показываем в этой статье, межпотоковое вмешательство может разрушить параллелизм доступа на уровне банков к отдельным потокам. Запросы памяти, задержки которых в противном случае были бы в значительной степени перекрыты, эффективно сериализуются, что может значительно снизить производительность потока. Кроме того, некоторые потоки могут иметь несправедливый приоритет, в то время как другие - возможно, более важные - потоки могут голодать в течение длительных периодов времени.

Такие негативные эффекты неконтролируемых межпотоковых помех в системе памяти DRAM являются критическими препятствиями для создания жизнеспособных, масштабируемых и управляемых систем CMP, поскольку они могут привести к 1) низкой производительности системы и уязвимости для отказа в обслуживании [22, 41], 2) непредсказуемая производительность программы, что делает анализ производительности, оптимизацию и изоляцию чрезвычайно трудными [28, 22, 25], 3) значительный дискомфорт для конечного пользователя, который, естественно, ожидает, что потоки с более высокими (равными) приоритетами получат более высокие (равные) доли производительности системы. По мере увеличения числа ядер в микросхеме нагрузка на систему DRAM также будет увеличиваться, и производительность и справедливость, обеспечиваемые системой DRAM, станут критическими определяющими факторами производительности будущих платформ CMP. Следовательно, для обеспечения жизнеспособных, масштабируемых и предсказуемых систем CMP необходимы справедливые и высокопроизводительные методы планирования доступа к памяти, которые контролируют и минимизируют межпотоковые помехи [28, 22, 25].

В этой статье мы предлагаем новый подход к обеспечению справедливого и высокопроизводительного планирования DRAM. Наш алгоритм планирования, называемый пакетным планированием с учетом параллелизма (parallelism-aware batch scheduling, PAR-BS), основан на двух новых ключевых идеях: пакетное планирование запросов и планирование DRAM с учетом параллелизма. Во-первых, PAR-BS работает, группируя ограниченное количество запросов DRAM в пакеты на основе времени их поступления и запрашивающих потоков. Запросы из самой старой партии имеют приоритет и поэтому гарантированно будут обслужены раньше других запросов. Таким образом, PAR-BS является справедливым и не требует голодания: он предотвращает голодание любого потока в системе DRAM из-за помех от других потенциально агрессивных потоков. Во-вторых, в пакете запросов PAR-BS учитывается параллелизм: она стремится сохранить параллелизм доступа на уровне банка (т. е. степень, в которой запросы DRAM потока обслуживаются параллельно в разных банках DRAM) каждого потока при наличии помех от запросов других потоков. Это делается путем попытки сгруппировать запросы из потока и обслуживать их одновременно (при условии, что они обращаются к разным банкам), используя эвристические правила определения приоритетов. Таким образом, наш подход уменьшает сериализацию запросов потока, которые в противном случае обслуживались бы параллельно, если бы поток выполнялся один в системе памяти.

Мы показываем, что компонент пакетной обработки запросов в PAR-BS является общей структурой, которая обеспечивает справедливость и свободу от голода при наличии межпоточных помех. В пакете запросов DRAM может быть реализован любой существующий и будущий алгоритм планирования доступа к DRAM (например, предложенный в [44, 33, 32, 28, 25]). Однако наши результаты показывают, что использование предложенного нами алгоритма планирования с учетом параллелизма обеспечивает лучшую справедливость, а также пропускную способность системы. Мы описываем, как PAR-BS работает в пакете, и анализируем сложные компромиссы, связанные с пакетным и параллельным планированием вредоносного ПО, с точки зрения справедливости, пропускной способности DRAM, использования локальности буфера строк и параллелизма доступа на уровне банков для отдельных потоков. Мы также опишем, как системное программное обеспечение может управлять PAR-BS для обеспечения приоритетов потоков и изменения уровня требуемой справедливости в системе DRAM.

Наши эксперименты сравнивают качественно и количественно PAR-BS с четырьмя ранее предложенными методами планирования DRAM, включая недавно предложенные планировщики доступа к памяти на основе справедливой очереди на основе QoS [28] (NFQ) и Stall-Time Fair [25] (STFM), а также общепринятый планировщик «первым пришел - первым обслужен» (FRFCFS) [44, 33]. Ни один из этих планировщиков не пытается сохранить параллельность отдельных потоков на уровне банков или строго гарантировать кратковременную свободу голодания при наличии межпотоковых помех. Наши результаты по очень широкому спектру рабочих нагрузок и конфигураций системы CMP показывают, что PAR-BS обеспечивает лучшую справедливость и пропускную способность системы.

**Вклады:** Мы делаем следующие вклады в этом документе:

• Мы показываем, что межпотоковые помехи могут разрушить параллелизм отдельных потоков на уровне банков, что приведет к значительному снижению пропускной способности системы. Мы представляем новую политику планирования DRAM с учетом параллелизма, которая поддерживает параллелизм отдельных потоков на уровне банка, а также учитывает локальность буфера строк.

• Мы представляем концепцию пакетирования запросов в общих планировщиках DRAM в качестве общей структуры, обеспечивающей справедливость/QoS для потоков и свободу голодания для запросов DRAM. Мы показываем, что пакетирование запросов является ортогональным и может использоваться с существующими алгоритмами планирования доступа DRAM, но оно наиболее выгодно при применении с планированием с учетом параллелизма. Мы описываем, как системное программное обеспечение может управлять гибкой основой справедливости, предоставляемой пакетной обработкой запросов, для обеспечения приоритетов потоков и контроля несправедливости в системе DRAM.

• Мы качественно и количественно сравниваем наш планировщик с четырьмя ранее предложенными планировщиками, ориентированными на справедливость или пропускную способность, и показываем, что PAR-BS обеспечивает как лучшую справедливость, так и лучшую пропускную способность системы. Наше предложение также проще реализовать, чем лучший ранее предложенный планировщик доступа к памяти, Stall-Time Fair Memory Scheduler [25], поскольку он не требует сложных вычислений, таких как деление.

1. **Мотивация**

Запросы DRAM - это операции с очень большой задержкой, которые сильно влияют на производительность современных процессоров. Когда инструкция загрузки отсутствует в кэш-памяти последнего уровня и нуждается в доступе к DRAM, процессор не может зафиксировать эту (и любую последующую) инструкцию, потому что инструкции фиксируются в программном порядке для поддержки точных исключений [35]. Окно инструкций процессора заполняется через несколько циклов после пропуска кэша последнего уровня [13, 24], и процессор останавливается до тех пор, пока ошибка не будет обработана DRAM. Современные процессоры пытаются уменьшить потерю производительности из-за доступа DRAM, обслуживая другие обращения к DRAM параллельно с ним. Такие методы, как выполнение вне очереди [40], неблокирующие кэши [15] и выполнение циклов выполнения [5, 23], стремятся перекрыть задержку будущих обращений DRAM с текущим доступом так, что процессору не нужно останавливаться (на долго) для будущих DRAM доступов. Вместо этого, на абстрактном уровне, процессор останавливается один раз для всех перекрывающихся обращений, а не останавливается для каждого доступа в последовательном порядке [24]. Концепция параллельного генерирования и обслуживания множественного доступа DRAM называется параллелизмом уровня памяти (MLP) [9].

Эффективность вышеупомянутых методов допуска задержки зависит от того, действительно ли параллельные обращения к DRAM обслуживаются параллельно различными банками DRAM (т.е. поддерживается ли параллелизм на уровне потоков внутри банка). В одноядерной поток имеет эксклюзивный доступ к банкам DRAM, поэтому его параллельные обращения к DRAM обслуживаются параллельно, если они не относятся к одному и тому же банку. Это проиллюстрировано в простом концептуальном примере, показанном на . Задержка Request1 (Req1) скрыта задержкой Request0 (Req0), эффективно воздействуя только на задержку доступа к одному банку для ядра обработки потока. После обслуживания Req0 ядро может зафиксировать нагрузку 0 и, таким образом, включить декодирование / выполнение будущих инструкций. Когда Load 1 становится самой старой инструкцией в окне, ее промах уже был обслужен, и поэтому процессор может продолжить вычисления без остановки.

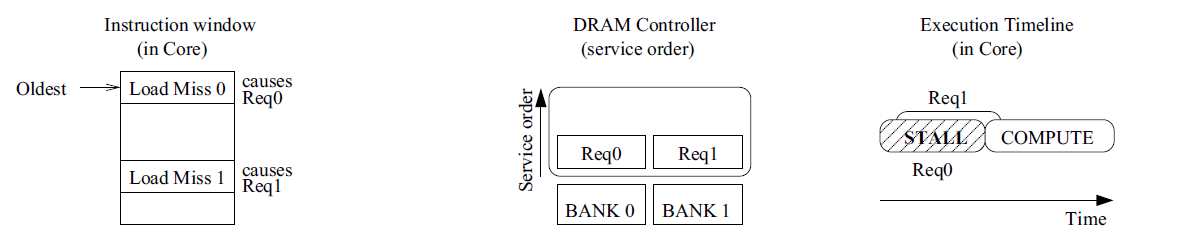


Рисунок 1 - Пример, показывающий, как задержки двух запросов DRAM перекрываются в одноядерной системе

К сожалению, если несколько потоков генерируют запросы памяти одновременно (например, в системе CMP), современные контроллеры DRAM планируют невыполненные запросы таким образом, что полностью игнорирует присущий параллелизм потоков на уровне памяти. Вместо этого современные контроллеры DRAM стремятся максимизировать пропускную способность данных DRAM, то есть количество запросов DRAM, обслуживаемых в секунду [44, 33, 32]. Как мы покажем в этой статье, слепая максимизация пропускной способности данных DRAM не минимизирует время простоя потока (которое напрямую коррелирует с пропускной способностью системы). Даже если пропускная способность DRAM может быть максимально увеличена, некоторые потоки могут слишком долго останавливаться, если контроллер DRAM разрушает их параллелизм на уровне банка и сериализует свои обращения к памяти вместо того, чтобы обслуживать их параллельно.

Пример на рисунке 2 иллюстрирует, как отсутствие параллелизма может привести к неоптимальной пропускной способности системы CMP и увеличенным задержкам. Мы предполагаем два ядра, каждое из которых работает с одним потоком: поток 0 (T0) и поток 1 (T1). Каждый поток имеет два одновременных запроса DRAM, вызванных последовательными независимыми пропусками загрузки (Load 0 и Load 1), и запросы направляются в два разных банка DRAM. На рисунке показано 1) (вверху), как текущий планировщик DRAM может разрушить параллелизм банка между потоками, тем самым увеличивая время задержки потока, и 2) (внизу), как планировщик с поддержкой параллелизма может планировать запросы более эффективно.

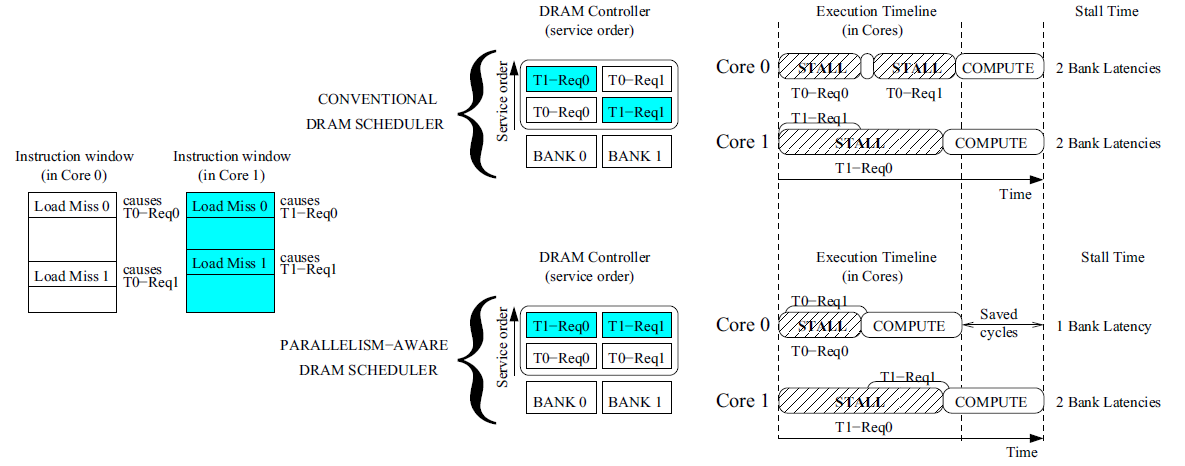


Рисунок 2 - Концептуальный пример, показывающий важность включения понимания параллелизма в решениях планирования DRAM

С помощью обычного планировщика DRAM без параллелизма (такого как любой ранее предложенный планировщик [44, 33, 32, 28, 25]) запросы могут обслуживаться в порядке их поступления, показанном на рисунке 2 (вверху). Во-первых, запрос T0 к банку 0 обслуживается параллельно с запросом T1 к банку 1. Позже, запрос T1 к банку 0 обрабатывается параллельно с запросом T0 к банку 1. Этот сервисный заказ сериализует параллельные запросы каждого потока и, следовательно, выставляет две задержки доступа к банку каждому ядру. Как показано на временной шкале выполнения (вверху справа), вместо двух однократных остановок (т. е. для одной задержки доступа к банку) для двух запросов оба ядра останавливаются дважды. Core 0 сначала останавливается для Load 0, а вскоре после этого и для Load 1. Core 1 останавливается для Load 0 для двух задержек доступа к банку.

Напротив, планировщик с поддержкой параллелизма обслуживает параллельные запросы каждого потока параллельно, что приводит к порядку обслуживания и временной шкале выполнения, показанным на рисунке 2 (внизу). Планировщик сохраняет параллелизм банка, сначала планируя запросы T0 параллельно, а затем запросы T1. Это позволяет Core 0 выполняться быстрее (на рисунке это обозначено как «Сохраненные циклы»), поскольку оно останавливается только для одной задержки доступа к банку. Время простоя Core 1 остается неизменным: хотя его второй запрос (T1-Req1) обслуживается позже, чем с обычным планировщиком, T1-Req0 все еще скрывает задержку T1-Req1.

Важным наблюдением является то, что планирование запросов с учетом параллелизма улучшает общую пропускную способность системы, потому что одно ядро теперь выполняется намного быстрее: среднее время задержки ядра составляет 1,5 задержки доступа к банку с планировщиком с учетом параллелизма (снизу), тогда как с обычным планировщиком оно равно 2 (сверху) Хотя в этом примере для простоты показаны только два ядра, разрушение параллелизма между потоками становится все хуже, так как все больше ядер совместно используют систему DRAM.

**Наша цель**: наша цель в этой статье - разработать честный планировщик памяти с поддержкой QoS, обеспечивающий высокую пропускную способность системы. Основываясь на наблюдении, что межпотоковые помехи разрушают параллелизм на уровне банков потоков, одновременно работающих на CMP, и, следовательно, ухудшают пропускную способность системы, мы учитываем параллелизм в дизайне нашего справедливого и высокопроизводительного планировщика доступа к памяти. С этой целью мы разрабатываем ключевые понятия пакетной обработки запросов и определения приоритетов запросов с учетом параллелизма, которые подробно описаны в разделе 4.

1. **Справочная информация о контроллерах памяти DRAM**

В этом разделе дается краткое описание того, как работают современные системы и контроллеры SDRAM. Система DRAM представлена на уровне абстракции, достаточном для понимания терминологии и ключевых понятий этой статьи. За подробным описанием мы отсылаем читателя к [33, 4, 25].

Современная микросхема SDRAM состоит из нескольких банков DRAM, что позволяет параллельно осуществлять несколько обращений к памяти, если им требуются данные из разных банков. Каждый банк DRAM представляет собой двумерный массив, состоящий из столбцов и строк. Строки обычно хранят данные в последовательных ячейках памяти и имеют размер 1-2 КБ. Доступ к данным в банке возможен только из буфера строк, который может содержать не более одной строки. Банк содержит один буфер строк. Время, необходимое для обслуживания запроса DRAM, зависит от состояния буфера строк и подразделяется на три категории:

• **Попадание строки:** запрос к строке, которая в данный момент открыта в буфере строк. Контроллеру DRAM необходимо выдать только команду чтения или записи в банк DRAM, что приводит к задержке доступа к банку (см. Таблицу 2).

• **Строка закрыта**: в буфере строк нет открытой строки. Контроллеру DRAM необходимо сначала выполнить команду активации, чтобы открыть нужную строку, а затем команду чтения / записи, что приводит к задержке доступа к банку + .

• **Конфликт строк**: запрос к строке отличается от того, который в данный момент находится в буфере строк. Контроллеру DRAM необходимо сначала закрыть строку, выполнив команду предварительной зарядки, затем открыть нужную строку (активировать) и затем выполнить команду чтения / записи. Эти обращения влекут за собой наибольшую задержку доступа к банку + + .

Контроллер DRAM состоит из буфера запросов памяти, который буферизует запросы памяти (и их данные), пока они ожидают обслуживания, и планировщика (возможно, двухуровневого), который выбирает следующий запрос для обслуживания [33, 28, 25]. При выборе следующего запроса для обслуживания планировщик учитывает состояние банков DRAM и шин DRAM, а также состояние запроса. Команда DRAM для запроса может быть запланирована только в том случае, если ее планирование не вызывает конфликты ресурсов (банк и адрес/данные/командная шина) и не нарушает какие-либо временные ограничения DRAM. Говорят, что такая команда DRAM готова.

Из-за большого несоответствия в задержке, вызванной доступом к строке и доступом с конфликтом строк / закрытым доступом, современные контроллеры DRAM используют методы планирования, которые устанавливают приоритеты запросов на попадание строки по сравнению с другими запросами, включая более молодые. Современный контроллер памяти использует политику планирования FR-FCFS (first-ready first-come-first-serve) [44, 33, 32], которая отдает приоритет готовым командам DRAM из 1) запросов на попадание строки по сравнению с другими и 2) статус попадания строки равен более старым запросам, чем более молодым. Такая политика планирования имеет целью минимизировать среднюю задержку обслуживания запросов DRAM и, таким образом, максимизировать пропускную способность данных, полученных от DRAM. Для однопоточных систем показано, что политика FR-FCFS обеспечивает лучшую среднюю производительность [33, 32], значительно лучше, чем более простая политика FCFS, которая просто планирует все запросы в соответствии с порядком их поступления, независимо от состояния буфера строки.

Когда несколько потоков совместно используют систему DRAM, политика планирования FR-FCFS имеет тенденцию неоправданно расставлять приоритеты для потоков с высокой локализацией буфера строк (т. е. частоты обращений к буферу строк) по сравнению с потоками с относительно низкой локализацией буфера строк из-за правила определения приоритетов строка-попадания-первая (row-hit-first). Он также имеет тенденцию неоправданно устанавливать приоритеты для потоков, интенсивно использующих память, по сравнению с неинтенсивными из-за самого старого правила определения . В результате, хотя FR-FCFS достигает высокой пропускной способности данных DRAM, он может истощать запросы/потоки в течение длительных периодов времени, вызывая несправедливость и относительно низкую общую пропускную способность системы [28, 22, 25].

Предыдущие исследования [28, 22, 25] экспериментально продемонстрировали несправедливость FR-FCFS и предложили новые политики планирования, которые более справедливы и которые обеспечивают QoS для различных потоков. Nesbit и соавт. [28] применили методы Network Fair-Queuing (NFQ) к контроллерам DRAM, чтобы разделить полосу пропускания DRAM между несколькими потоками, совместно использующими систему DRAM. Mutlu и Moscibroda [25] предложили планировщик справедливой памяти в режиме ожидания (STFM), целью которого является выравнивание замедлений, испытываемых потоками, по сравнению с тем, когда каждый из них запускается один. Ни одна из этих предыдущих политик планирования не учитывает внутри потоковый параллелизм банков, который, как видно из Раздела 2, может значительно снизить производительность системы, когда запросы разных потоков вмешиваются в систему DRAM.

1. **Параллелизм с учетом пакетного планирования**

**Обзор**: Наш предложенный алгоритм планирования DRAM разработан, чтобы обеспечить 1) настраиваемый субстрат для справедливости и качества обслуживания и QoS и 2) высокую пропускную способность системы CMP путем включения понимания параллелизма в решениях планирования. Для достижения этих целей пакетное планирование с учетом параллелизма (PAR-BS) состоит из двух компонентов. Первый компонент представляет собой компонент пакетной обработки запросов (BS) или просто пакетный компонент, который группирует несколько невыполненных запросов DRAM в пакет и гарантирует, что все запросы, принадлежащие текущему пакету, обслуживаются до формирования следующего пакета. Пакетная обработка не только обеспечивает справедливость, но также обеспечивает удобную гранулярность (то есть пакетную обработку), в рамках которой возможно выполнение недобросовестных, но высокопроизводительных оптимизаций планирования команд DRAM. Второй компонент нашего предложения, параллельное планирование в рамках пакетного планирования (PAR), направлен на уменьшение среднего времени остановки потоков в пакете (и, следовательно, на увеличение пропускной способности CMP), пытаясь обслуживать запросы каждого потока параллельно в банках DRAM.

После отдельного описания двух компонентов мы обсудим преимущества / недостатки нашего предложения по сравнению с существующими планировщиками DRAM и представим возможные альтернативные варианты проектирования в разделах 4.3 и 4.4 соответственно.

* 1. **Запрос пакетирования**

Идея пакетной обработки состоит в том, чтобы последовательно группировать невыполненные запросы в буфере запросов памяти в большие блоки, называемые пакетами. Планировщик DRAM избегает переупорядочения запросов между пакетами, устанавливая приоритеты запросов, принадлежащих текущему пакету, над другими запросами. Как только все запросы пакета обслуживаются (т.е. когда пакет завершен), формируется новый пакет, состоящий из невыполненных запросов в буфере запросов памяти, которые не были включены в последний пакет. Таким образом, группируя запросы в более крупные единицы в соответствии с временем их поступления, пакетирование, в отличие от FR-FCFS и других существующих схем, предотвращает голодание запросов при очень тонкой детализации и обеспечивает устойчивый и справедливый прогресс во всех потоках. В то же время формирование пакетов позволяет гибко переупорядочивать запросы в пакете, чтобы максимально использовать локальность буфера строк и банковский параллелизм без существенного нарушения справедливости потока.

Компонент дозирования (BS) PAR-BS работает следующим образом. Каждый запрос в буфере запроса памяти имеет связанный бит, указывающий, принадлежит ли запрос текущему пакету. Если запрос принадлежит текущему пакету, этот бит установлен, и мы называем запрос отмеченным. BS формирует партии, используя положения, показанные в правиле 1.

**Правило 1** Планировщик PAR-BS: формирование партии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1: **Формирование нового пакета**: новый пакет формируется, когда в буфере запросов памяти не осталось помеченных запросов, то есть, когда все запросы из предыдущего пакета были полностью обслужены.

2: **Маркировка**: при формировании новой партии BS оценивает невыполненные запросы Marking-Cap на банк для каждого потока; эти запросы формируют новую партию.

Marking-Cap - это системный параметр, который ограничивает количество запросов, выданных потоком для определенного банка, которые могут быть частью пакета. Например, если Marking-Cap равен 5, а поток имеет 7 невыполненных запросов на банк, PAR-BS отмечает только 5 самых старых из них. Если Marking-Cap не установлен, все ожидающие запросы помечаются при формировании новой партии.

PAR-BS всегда отдает приоритет отмеченным запросам (то есть запросам, относящимся к текущей партии), а не неотмеченным запросам в данном банке. С другой стороны, PAR-BS не тратит впустую полосу пропускания и не излишне задерживает запросы: если нет отмеченных запросов к данному банку, то к этому банку планируются невыполненные неотмеченные запросы. Для выбора между двумя отмеченными или двумя неотмеченными запросами может использоваться любой существующий или новый алгоритм планирования DRAM (например, FR-FCFS). В PAR-BS этим компонентом планирования внутри партии является PAR, который мы опишем далее.

* 1. **Параллелизм с внутрипартийным планированием**

Пакетирование, естественно, обеспечивает удобную гранулярность (то есть пакет), в рамках которой планировщик может оптимизировать решения по планированию для получения высокой производительности. Эта оптимизация должна преследовать две основные цели. Он должен одновременно максимизировать 1) локальность буфера строк и 2) параллелизм потока внутри пакета. Первая цель важна, потому что, если в пакете поддерживается высокая частота обращений к строке, доступ к банкам в среднем будет меньше задержек, что увеличивает пропускную способность системы DRAM. Вторая цель также важна, потому что параллельное планирование нескольких запросов от потока к разным банкам эффективно сокращает время ожидания этого потока. К сожалению, как правило, трудно одновременно достичь этих целей, например, FR-FCFS жертвует параллелизмом вместо буфера .

Наш алгоритм внутрипартийного планирования, PAR, использует правила приоритезации запросов, показанные в правиле 2, чтобы использовать как локальность буфера строк, так и параллелизм банка. Внутри пакета запросы на попадание в строку имеют приоритет. Это увеличивает локальность буфера строк и гарантирует, что любые строки, которые были оставлены открытыми по запросам предыдущего пакета, будут наилучшим образом использованы в следующем пакете. Во-вторых, запросы от потоков с более высоким рангом имеют приоритет над запросами от потоков с более низким рангом для увеличения параллелизма на уровне банка, как подробно объяснено ниже. Наконец, при прочих равных, более старый запрос имеет приоритет над более молодым.

**Правило 2** Планировщик PAR-BS: расстановка приоритетов запросов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1: **BS - Marked-запросы-сначала**: помеченные готовые запросы имеют приоритет над запросами, которые не помечены.

2: **RH – строка-попадание-сначала**: запросы на попадание в строку имеют приоритет над запросами на конфликтные/закрытые строки.

3: **RANK** - **сначала более высокий ранг**: запросы от потоков с более высоким ранжированием имеют приоритет над запросами от потоков с более низким рангом.

4: **FCFS - самые старые**: более старые запросы имеют приоритет над более молодыми.

**Ранжирование потоков:** PAR-BS использует схему приоритетов потоков на основе ранга в пакете, чтобы максимизировать внутри поточный параллелизм банков при сохранении локализации буфера строк. Когда формируется новый пакет, планировщик DRAM вычисляет ранжирование среди всех потоков, у которых есть запросы в пакете. Во время обработки пакета вычисленное ранжирование остается тем же, и запросы от потоков с более высоким рейтингом имеют приоритет над запросами от потоков с более низким рейтингом. Эффект планирования на основе ранжирования заключается в том, что различные потоки располагаются по приоритетам в одном и том же порядке во всех банках, и, таким образом, запросы каждого потока, скорее всего, будут обслуживаться параллельно всеми банками.

**Как ранжировать потоки в пакете:** хотя концептуально любая основанная на ранжировании схема усиливает внутрипартийный внутри поточный параллелизм банков, конкретная процедура ранжирования оказывает существенное влияние на пропускную способность и честность CMP.

Хорошая схема ранжирования должна эффективно различать потоки, интенсивно использующие память, и потоки, не интенсивные (и потоки с высоким параллелизмом банков). Если не интенсивный поток с несколькими запросами ранжируется ниже, чем интенсивный поток, его запросы могут быть чрезмерно задержаны в пакете. Как объяснено в [25], справедливый планировщик DRAM должен выровнять замедление, связанное с DRAM каждого потока, по сравнению с тем, когда поток работает один в той же системе памяти. Поскольку не интенсивный поток или поток с высоким параллелизмом банков по своей природе имеет низкое время задержки, связанное с DRAM, при запуске в одиночку, задержка его запросов в пакете приводит к гораздо большему замедлению, чем это было бы для интенсивного потока, у которого время задержки DRAM уже велико, даже когда он запущен в одиночку. Чтобы избежать этой несправедливости (и потери пропускной способности системы, как объясняется ниже), наша схема ранжирования основана на принципе «самое короткое задание» [36]: она ранжирует неинтенсивные потоки выше, чем интенсивные.

Помимо справедливости, основное обоснование первого принципа самого короткого задания заключается в том, что оно имеет тенденцию уменьшать среднее время завершения пакета для потоков внутри . Время завершения пакета потока - это время между началом пакета и временем обслуживания последнего отмеченного запроса потока от пакета. Это напрямую соответствует времени задержки потока, связанного с памятью в пакете. Сокращая среднее время завершения пакета, планирование первоочередного задания с кратчайшим сроком работы улучшает общую пропускную способность системы, поскольку потоки в среднем меньше останавливаются для запросов DRAM, тем самым ускоряя выполнение их потоков команд.

Конкретно, PAR-BS использует схему ранжирования Max-Total, показанную в правиле 3, для вычисления ранга каждого потока в пакете.

Правило 3 Планировщик PAR-BS: Рейтинг темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1: **Правило Max:** для каждого потока планировщик находит максимальное количество отмеченных запросов к любому банку, называемое максимальной нагрузкой банка (max-bank-load). Поток с более низкой максимальной нагрузкой банка ранжируется выше, чем поток с более высокой максимальной нагрузкой банка.

2: **Общее правило Tie-breaker**: для каждого потока планировщик отслеживает общее количество отмеченных запросов, называемых общей нагрузкой. Если потоки ранжируются одинаково в соответствии с правилом Max, поток с меньшей общей загрузкой оценивается выше, чем поток с более высокой общей нагрузкой. Любые оставшиеся связи разрываются случайным образом.

Максимальное количество невыполненных запросов к любому банку соотносится с «нехваткой задания», то есть с минимальной задержкой памяти, которая требуется для обслуживания всех запросов из потока, если они обрабатывались полностью параллельно. Поток с высоким рейтингом имеет несколько помеченных запросов, отправляемых в один и тот же банк, и, следовательно, может быть быстро завершен. Распределяя приоритеты запросов от таких потоков с высоким рейтингом в пакете, PAR-BS гарантирует, что неинтенсивные потоки или потоки с высоким параллелизмом банков будут быстро продвигаться и не будут долго задерживаться без необходимости.

**Пример**: на рисунке 3 показан пример, который дает представление о том, почему предлагаемая нами схема приоритетов внутри партии сохраняет внутри поточный параллелизм и повышает пропускную способность системы. Рисунок отвлекает многие детали планирования , но обеспечивает основу для понимания параллелизма и компромиссов локальности. Мы принимаем задержку, равную 1 для запросов на конфликт строк и 0,5 для запросов на попадание строк. На рис. 3 (а) показан порядок поступления запросов в каждом банке, что эквивалентно их порядку обслуживания с планировщиком FCFS. FCFS не использует локальность и не сохраняет внутри потоковый параллелизм банка и, следовательно, приводит к наибольшему среднему времени завершения из четырех потоков (5 единиц задержки). FR-FCFS максимизирует частоту обращений к буферу строк, переупорядочивая запросы на попадание строк по сравнению с другими, но, как показано на рисунке 3 (b), он не сохраняет внутри потокового банковского параллелизма. Например, хотя поток 1 имеет только три запроса, предназначенных для разных банков, FR-FCFS обслуживает все три запроса последовательно. В зависимости от истории запросов памяти расписание, показанное на рис. 3 (b) для FR-FCFS, также является возможным сценарием выполнения при использовании планировщиков NFQ [28] или STFM [25] с учетом QoS, поскольку эти планировщики не знают о внутреннем банковском параллелизме.

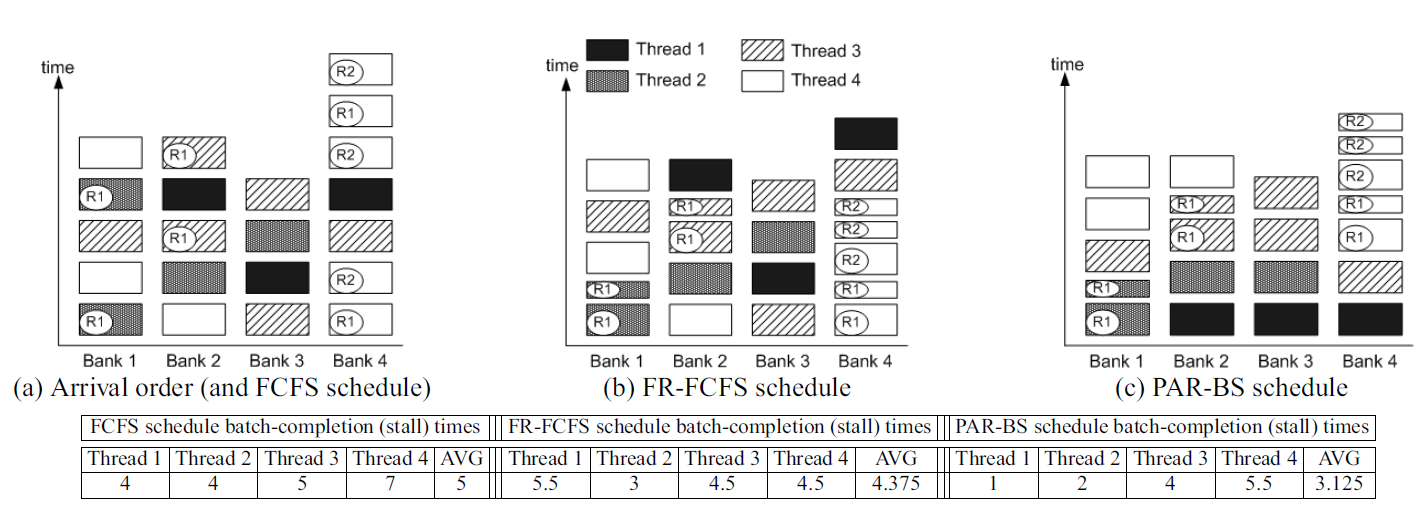


Рисунок 3 - Упрощенная абстракция планирования в пакете, содержащем запросы из 4 потоков. Прямоугольники представляют помеченные запросы из разных потоков; Самые нижние запросы - самые старые запросы для банка. Те запросы, которые влияют или приводят к попаданию в строку, помечаются номером строки, к которой они обращаются; если два запроса к одной и той же строке обслуживаются последовательно, второй запрос является попаданием в строку с меньшей задержкой доступа. Предполагается, что первый запрос к каждому банку является конфликтом строк.

Рисунок 3 (с) показывает, как PAR работает в партии. Поток 1 имеет не более одного запроса на банк (что приводит к наименьшей максимальной загрузке банка, равной 1), и поэтому он занимает самое высокое место в этом пакете. В обоих потоках 2 и 3 максимальная загрузка банка равна двум, но, поскольку общее количество запросов в потоке 2 меньше, он оценивается выше потока 3. Поток 4 занимает наименьшее значение, поскольку максимальная загрузка банка равна 5. Так как поток 1 занимает первое место, его три запроса планируются идеально параллельно, перед другими запросами. Точно так же запросы потока 2 планируются максимально параллельно. В результате PAR максимизирует параллельность банков неинтенсивных потоков и завершает их запросы как можно быстрее, позволяя соответствующим ядрам быстро прогрессировать. По сравнению с FR-FCFS или FCFS, PAR значительно ускоряет потоки 1, 2 и 3, но существенно не замедляет поток 4. Среднее время завершения сокращается до 3,125 единиц задержки.

Обратите внимание, что в дополнение к хорошему параллелизму банков наше предложение обеспечивает такую же локальность буфера строк, что и FR-FCFS в пакете, потому что в пакете PAR-BS всегда отдает приоритет отмеченным запросам попадания строк по сравнению с запросами конфликта с.

* 1. **Преимущества, недостатки, компромиссы**

Компонент запроса пакетирования нашего предложения имеет следующие основные преимущества:

• **Честность и предотвращение голодания**: пакетная обработка гарантирует отсутствие кратковременного или долгосрочного голодания: каждый поток может добиться прогресса в каждом пакете, независимо от шаблонов доступа к памяти других . Число запросов от потока, запланированных до запросов другого потока строго ограничен размером партии. Помимо FCFS, ни один из существующих планировщиков не предоставляет аналогичного понятия предотвращения голода. В FR-FCFS поток с интенсивной памятью с отличной локализацией буфера строк может захватывать банк в течение длительного времени, если он может выдавать большое количество запросов на попадание строки в один и тот же банк подряд. В зависимости от истории моделей доступа, кратковременное голодание также возможно в NFQ и STFM, особенно из-за проблем с бездействием и балансом доступа к банку [25], связанных с NFQ, и неточных оценок замедления в STFM [25]. В PAR-BS потоки, интенсивно использующие память, не могут задерживать запросы от неинтенсивных в течение длительного времени.

• **Субстрат для использования параллелизма банков**: пакетирование позволяет использовать высокоэффективные политики внутрипартийного планирования (такие как PAR). Без пакетов (или любого другого аналогичного представления о группах запросов во времени) разработка планировщика с учетом параллелизма является сложной, так как неясно, в каком контексте следует оптимизировать параллелизм банков.

**• Гибкость и простота реализации**. Несмотря на то, что идея пакетирования наиболее полезна в сочетании с PAR, ее можно использовать в сочетании с любой существующей или будущей политикой планирования команд DRAM. Таким образом, пакетирование представляет собой простую и гибкую структуру, которая может использоваться для повышения справедливости существующих алгоритмов планирования. Мы исследуем производительность и справедливость использования политик FCFS и FR-FCFS в пакете в Разделе 8.3.3.

Возможным недостатком нашей схемы является то, что она требует тщательного определения Marking-Cap. Если Marking-Cap большой, PAR-BS может страдать от таких же проблем несправедливости, как FR-FCFS, хотя и не в такой степени. Если поток без большого объема памяти выдает запрос, который просто пропускает формирование нового пакета, запрос должен ждать, пока все запросы от текущего пакета к одному и тому же банку не будут обслужены, что замедляет поток не интенсивного потока. С другой стороны, небольшая Marking-Cap может замедлять потоки, интенсивно использующие память, поскольку большинство запросов Marking-Cap на поток и на банк включаются в пакет, а остальные откладываются до следующего пакета. Есть еще один важный недостаток, связанный с наличием небольших пакетов: чем ниже Marking-Cap, тем ниже частота попадания в буфер строк для потоков с высокой внутренней локализацией буфера строк. За пределами пакета отмеченный запрос конфликта строк имеет приоритет над не отмеченным запросом попадания строки. Чем меньше пакеты (чем меньше размер разметки), тем чаще может быть нарушен поток обращений к строке, что увеличивает время доступа к запросам. В разделе 8.3.1 подробно анализируется компромисс между справедливостью и производительностью различных настроек Marking-Cap.

**Параллелизм в пакетном планировании** одновременно обеспечивает высокую степень параллелизма банков и локальности буфера строк, как описано в предыдущем разделе. Ни одна из известных нам схем планирования DRAM не оптимизирует параллелизм внутри потоков. В соответствии с общей теорией планирования машин [36], использование схемы ранжирования Max-Total для определения приоритетов потоков с меньшим количеством запросов уменьшает среднее время остановки потоков в пакете. Хотя может показаться, что этот принцип «самая короткая работа в первую очередь» несправедливо наказывает за интенсивные потоки памяти, наши экспериментальные оценки в разделе 8 показывают, что этот эффект несущественен. Есть две причины: 1) перекрывающая схема пакетирования обеспечивает высокую степень справедливости, 2) задержка потока, интенсивно использующего память, приводит к относительно меньшему замедлению, поскольку время задержки интенсивного потока, связанное с DRAM, выше. Поэтому внутри пакета планировщик должен свободно оптимизировать работу для сокращения времени простоя, завершая потоки с небольшим количеством параллельных запросов, как можно быстрее.

* 1. **Альтернативы дизайна**

Мы экспериментировали с различными новыми, альтернативными схемами пакетного планирования и планирования внутри партии. Кратко опишем эти схемы для полноты. Наши оценки в разделе 8 показывают, что в среднем по большому и разнообразному набору рабочих нагрузок эти альтернативные схемы работают хуже, чем наша схема PAR-BS.

Способ пакетирования в PAR-BS можно назвать полным пакетированием, поскольку он требует, чтобы пакет запросов был завершен полностью до того, как будет запущен следующий пакет. Существуют альтернативные способы пакетирования.

**Статическое пакетирование на основе времени**. В этом подходе ожидающие запросы периодически помечаются с использованием статического интервала времени, независимо от того, завершен или нет предыдущий пакет. Схема характеризуется системным параметром Batch-Duration, который описывает, через какой интервал времени формируется новая партия. В начале новой партии немаркированные запросы помечаются с учетом Marking-Cap, в то время как запросы, уже помеченные из предыдущей партии, остаются таковыми. По сравнению с PAR-BS, этот подход к пакетированию не обеспечивает строгих гарантий предотвращения голода и может привести к значительной несправедливости, как показано в разделе 8.3.2.

**Пакетная обработка пустого слота (Eslot)**: если в PAR-BS запрос поступает в систему DRAM немного позже формирования новой партии, он может быть отложен до начала новой партии, что приводит к большому времени задержки, особенно для неинтенсивного потока. Пакетная обработка пустых слотов пытается облегчить эту проблему, позволяя добавлять запросы в текущий пакет, если в пакете было отмечено меньше запросов Marking-Cap из этого потока для определенного банка. Другими словами, если во время формирования нового пакета поток не использует всю свою выделенную долю отмеченных запросов (т. е. Имеет «пустые слоты») в пакете, ему разрешается добавлять поздние запросы в пакет пока порог Marking-Cap не будет достигнут.

**Альтернативные политики внутрипартийного планирования**. Внутри пакета может использоваться много различных альтернативных методов приоритетов запросов/команд. Помимо ранжирования Max-Total, мы также оценили Total-Max (где порядок правила Max и правило Total поменялись местами), случайные и циклические схемы ранжирования. Кроме того, мы оценили использование FCFS и FR-FCFS в пакете - без какого-либо ранжирования - чтобы изолировать эффект понимания параллелизма в нашем предложении. В разделе 8.3.3 описаны компромиссы, связанные с альтернативными методами внутрипартийного планирования.

1. **Включение приоритетов потоков системного уровня**

До сих пор мы описывали PAR-BS, предполагая, что все потоки имеют одинаковый приоритет и, с точки зрения справедливости, должны испытывать одинаковые замедления, связанные с DRAM, при совместной работе. Однако системное программное обеспечение (операционная система или монитор виртуальной машины), вероятно, захочет назначить потокам приоритеты, чтобы показать, что некоторые потоки более / менее важны, чем другие. PAR-BS легко включает понятие приоритетов потоков для обеспечения поддержки системного программного обеспечения. Приоритет каждого потока передается PAR-BS с точки зрения уровней приоритетов 1, 2, 3,. , ., где уровень 1 обозначает наиболее важный поток (самый высокий приоритет), а большее число обозначает более низкий приоритет. Потоки с равным приоритетом должны быть в равной степени замедлены [25], но чем ниже приоритет потока, тем более допустимо его замедление. Мы настраиваем PAR-BS двумя способами, чтобы включить приоритеты потоков.

• **Маркировка на основе приоритетов**: запросы от потока с приоритетом X помечаются только для каждого X-го пакета. Например, запросы от потоков с наивысшим приоритетом с уровнем 1 помечаются в каждом пакете, запросы из потоков с уровнем 2 помечаются в каждом втором пакете и так далее. В противном случае механизм пакетирования остается тем же, то есть новый пакет формируется всякий раз, когда в буфере нет отмеченных запросов.

• **Внутри-пакетное планирование на основе приоритетов**: в правила приоритизации запросов внутри партии, показанные в правиле 2, добавлено дополнительное правило. Между правилами 1.BS --- Marked-запросы-сначала и 2.RH --- попадание-строки-сначала, мы добавляем новое правило PRIORITY --- Higher-priority-threads-first. То есть, учитывая выбор между двумя помеченными или двумя немаркированными запросами, PAR-BS устанавливает приоритет запроса от потока с более высоким приоритетом. Между запросами потоков с равным приоритетом другие правила приоритетов запросов остаются прежними.

Результатом этих двух изменений в PAR-BS является то, что потоки с более высоким приоритетом естественным образом планируются быстрее: они помечаются чаще и, таким образом, участвуют в большем количестве пакетов и имеют приоритет над другими запросами в пакете.

**Чисто оппортунистическая услуга**: В дополнение к целочисленным уровням приоритета PAR-BS предоставляет один конкретный уровень приоритета L, который указывает потоки с самым низким приоритетом. Запросы от таких потоков никогда не помечаются, и им присваивается самый низкий приоритет среди немаркированных запросов. Следовательно, запросы от потоков на уровне L планируются чисто условно - только по расписанию, если система памяти свободна - чтобы минимизировать их помехи в других потоках.

Наконец, мы предоставляем системному программному обеспечению возможность установить Marking-Cap, который служит рычагом для определения степени несправедливости в системе (см. Раздел 8.3.1).

1. **Стоимость внедрения и оборудования**

PAR-BS требует реализации пакетирования (правило 1) и политики приоритезации запросов, описанной в разделе 4.2 (правила 2 и 3). Современные контроллеры на основе FR-FCFS уже реализуют политики приоритетов. Каждому запросу DRAM назначается приоритет, и команда DRAM, принадлежащая запросу с наивысшим приоритетом, планируется среди всех готовых команд. PAR-BS расширяет приоритет каждого запроса DRAM, используя две дополнительные части информации: 1) помечен ли запрос или 2) ранг потока, которому принадлежит запрос (с использованием ранжирования Max-Total). Чтобы отслеживать эту дополнительную информацию, планировщику требуется дополнительное состояние, показанное в таблице 1. Предполагая 8-ядерный CMP, буфер запроса на 128 записей и 8 банков DRAM, дополнительное аппаратное состояние, включая Thread-ID, требуется для реализации PAR -BS (за пределами FR-FCFS) составляет 1412 бит.

Счетчики ReqsInBankPerThread и ReqsPerThread увеличиваются / уменьшаются, когда новый запрос входит / покидает буфер запроса памяти. Когда отмеченный запрос полностью обслуживается, контроллер DRAM уменьшает TotalMarkedRequests. Когда TotalMarkedRequests достигает нуля, контроллер запускает новый пакет с помощью 1) маркировки самых старых запросов Marking-Cap на банк от каждого потока, 2) вычисления нового Max-Total ранжирования потоков с использованием регистров ReqsInBankPerThread и ReqsPerThread. Таким образом, дополнительная логика, требуемая PAR-BS, состоит из логики, которая 1) маркирует запросы (логика маркировки), 2) определяет ранжирование потока (логика ранжирования) и 3) вычисляет приоритеты запросов на основе отмеченного состояния и ранга потока (логика приоритетов). И логика маркировки, и логика ранжирования используются только тогда, когда новая партия сформирована и реализована с использованием приоритетных кодеров, которые принимают в качестве входных данных соответствующую информацию в каждом случае. Логика приоритизации принимает в качестве входных данных отмеченный статус, статус попадания строки, ранг потока и идентификатор запроса для формирования единого значения приоритета (см. Рисунок 4) для каждого запроса в каждом цикле DRAM.

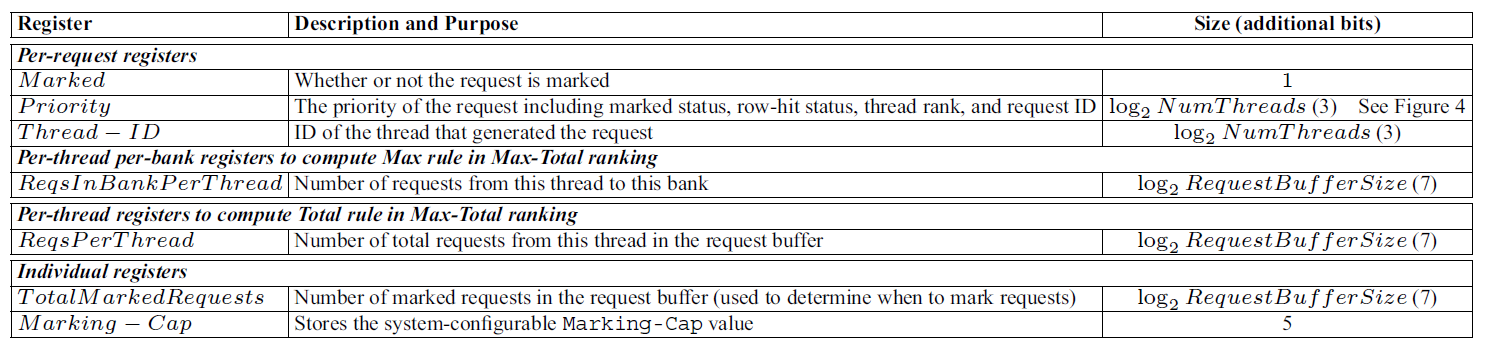


Таблица 1. Дополнительное состояние, необходимое для возможной реализации PAR-BS

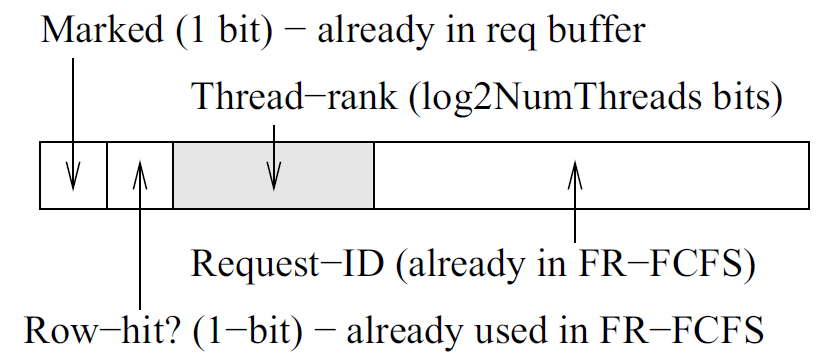


Рисунок 4 - Пример значения приоритета, назначенного каждому запросу. Рейтинг потока - единственное дополнительное хранилище, требуемое PAR-BS, поскольку все остальные поля либо требуются FR-FCFS, либо сохраняются вместе с запросом.

Обратите внимание, что ни одна из этой логики не находится на критическом пути процессора, потому что контроллер DRAM на кристалле работает на более высокой частоте, чем DRAM, и ему необходимо принимать решение о планировании только каждый цикл DRAM. Аналогичные политики приоритетов были реализованы в планировщиках команд, которые находятся на критическом пути. При необходимости логика маркировки / ранжирования может занять несколько циклов, поскольку маркировка / ранжирование выполняется только при формировании новой партии.

PAR-BS проще в реализации, чем предыдущий лучший планировщик STFM, для которого требуется существенная логика, включая делители, для оценки замедления потоков [25]. В отличие от STFM, PAR-BS основана только на простых правилах определения приоритетов, которые зависят от количества запросов и, следовательно, не требуют сложных арифметических операций.

1. **Экспериментальная методология**

Мы оцениваем наше предложение, используя циклический симулятор x86 CMP. Функциональный интерфейс симулятора основан на Pin [17] и iDNA [1]. Мы подробно моделируем систему памяти, точно отслеживая ограничения пропускной способности, конфликты и поддерживая конфликты между банком, портом, каналом и шиной. Таблица 2 показывает основные параметры DRAM и процессора. Мы масштабируем пропускную способность DRAM с учетом количества ядер. Наш расширенный технический отчет также оценивает различные параметры системы [26].

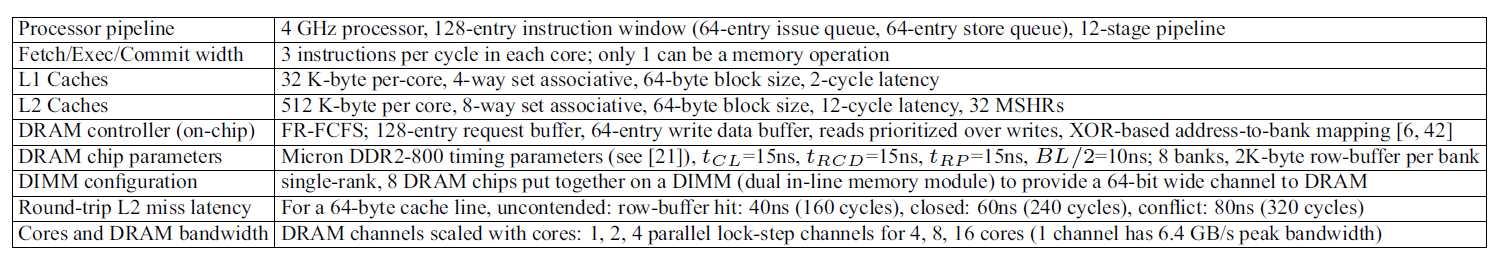


Таблица 2. Базовая конфигурация CMP и системы памяти

Для мы используем тесты SPEC CPU2006 и два настольных приложения Windows (Matlab и приложение синтаксического анализа xml). Каждый тест был скомпилирован с использованием gcc 4.1.2 с оптимизацией -O3 и запущен для 150 миллионов команд, выбранных из репрезентативной фазы выполнения [29].

Мы классифицируем тесты по восьми категориям на основе их интенсивности памяти (низкая или высокая), локальности буфера строк (низкая или высокая) и параллелизма на уровне банка (низкая или высокая). Мы определяем параллелизм на уровне банка (BLP) как среднее число запросов, обслуживаемых в банках DRAM, когда в банках DRAM обрабатывается хотя бы один запрос. Это определение следует определениям параллелизма на уровне памяти (MLP) Chou et al. [2]. Мы характеризуем поток на основе средней метрики времени ожидания для каждого запроса DRAM (AST / req), которая вычисляется путем деления числа циклов, в которых поток не может зафиксировать инструкции, поскольку самая старая инструкция является пропуском в DRAM на общее количество запросов загрузки DRAM, сгенерированных . В таблице 3 показаны характеристики системы и категории памяти в тестах, когда они выполняются отдельно в одном ядре базовой четырехъядерной CMP. Обратите внимание, что тесты с высоким уровнем BLP также имеют относительно низкую AST / req. На всех рисунках ориентиры упорядочены на основе их категории в таблице 3.

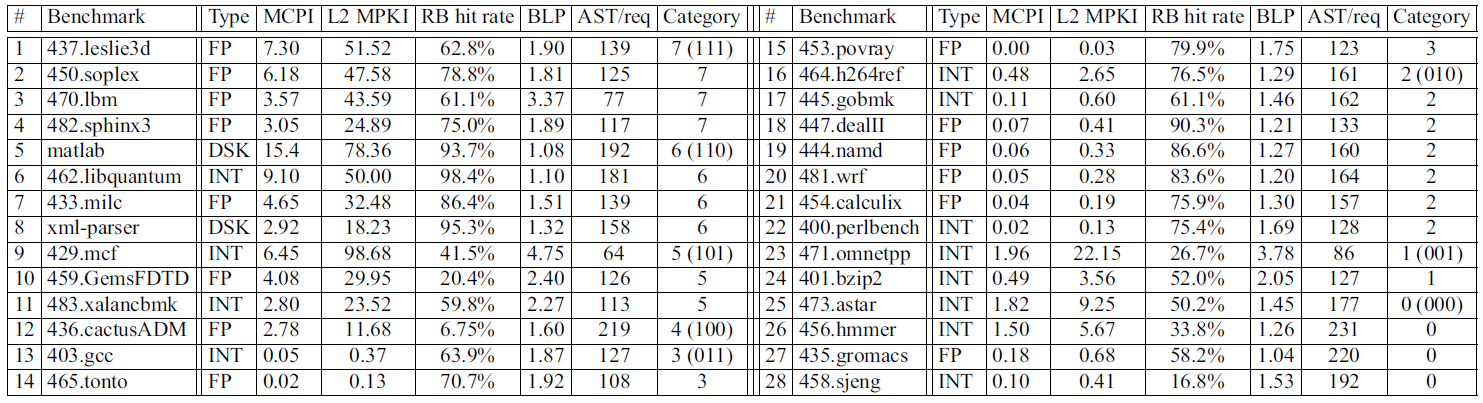
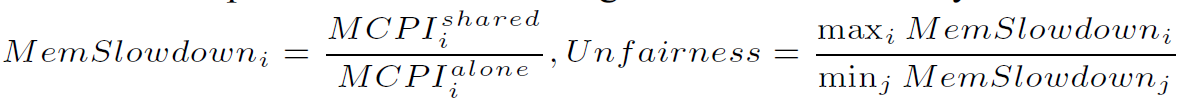


Таблица 3. Контрольные характеристики. MCPI: количество циклов памяти на инструкцию (количество циклов, потраченных на ожидание памяти, поделенное на количество инструкций), L2 MPKI: L2 промахов на 1000 инструкций, RB Hit Rate: частота обращений к буферу строк, BLP: параллелизм на уровне банка, AST / req: среднее время ожидания для каждого запроса DRAM. Категории определяются на основе MCPI (1: высокий, 0: низкий), частоты обращений к RB (1: высокий, 0: низкий) и BLP (1: высокий, 0: низкий).

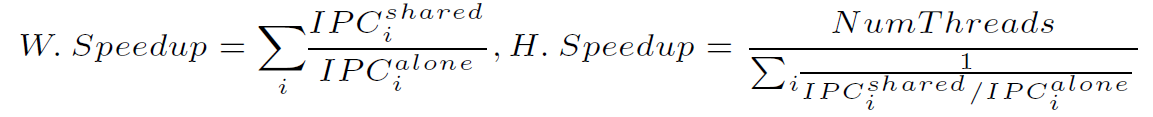
Мы оцениваем комбинации многопрограммных рабочих нагрузок, работающих на 4, 8 и 16-ядерных CMP. Для моделирования с четырьмя ядрами мы оценили 100 различных комбинаций, каждая из которых была сформирована путем псевдослучайного выбора эталона из каждой категории так, чтобы оценивались различные комбинации категорий. Для 8-ядерных симуляций мы оценили 16 различных комбинаций; и для 16-ядерных, 12 разных комбинаций. Ограничение пространства не позволяет нам перечислить все оцененные комбинации, но мы стараемся показать как можно больше результатов с репрезентативными индивидуальными комбинациями в Разделе 8.

* 1. **Метрики оценки**

Мы измеряем справедливость, используя индекс несправедливости, предложенный в [25,8 Это соотношение между максимальным замедлением, связанным с памятью, и минимальным замедлением, связанным с памятью, среди всех потоков, совместно использующих систему DRAM. Замедление потока, связанное с памятью, *i* - это время остановки памяти для каждой инструкции, которую он испытывает при работе вместе с другими потоками, поделенное на время остановки памяти для каждой инструкции, которую он испытывает, работая отдельно в той же системе:



Мы измеряем пропускную способность системы, используя Weighted-Speedup [37] и Hmean-Speedup [18], которые уравновешивают справедливость и пропускную способность [18]:



* 1. **Параметры оцененных схем**

Наш базовый контроллер использует политику планирования FR-FCFS. Все оцененные планировщики отдают приоритет запросам чтения DRAM над запросами записи DRAM, потому что запросы чтения могут напрямую блокировать продвижение вперед в ядрах обработки и, следовательно, более критичны для производительности. Если не указано иное, в наших экспериментах мы используем PAR-BS с Marking-Cap 5. При сравнении PAR-BS с другими планировщиками мы используем следующие параметры. **STFM**: Мы установили *α* = 1.10 и *IntervalLength* = , как предложено Мутлу и Москибродой [25]. **NFQ**: Мы используем лучшую схему Nesbit et al. (FQ-VFTF) [28], включая оптимизацию предотвращения приоритета с порогом .

1. **Результаты экспериментов**
   1. **Результаты по 4-ядерным системам**

Сначала мы проанализируем справедливость и пропускную способность PAR-BS по сравнению с ранее предложенными методами планирования DRAM, используя три тематических исследования 4-ядерных систем, которые подчеркивают типичное поведение различных алгоритмов планирования. Совокупные результаты более 100 рабочих нагрузок представлены в разделе 8.1.4.

* + 1. **Пример I: интенсивная память.**

Эта рабочая нагрузка включает в себя четыре теста с интенсивным использованием памяти, один с очень высоким параллелизмом на уровне банков (mcf). Рисунок 5 (слева) показывает замедление памяти каждого теста с различными планировщиками памяти. На рисунке 5 (справа) сравниваются пять различных планировщиков с точки зрения пропускной способности системы.

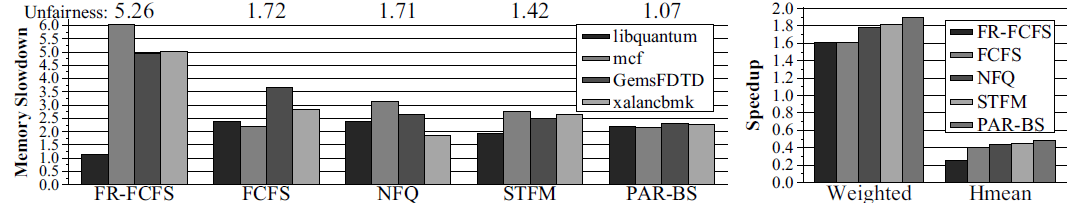


Рисунок 5 - 4-ядерная рабочая нагрузка с интенсивным использованием памяти: замедление работы памяти и несправедливость (слева), пропускная способность системы (справа)

• FR-FCFS и FCFS: обычно используемая политика планирования FR-FCFS очень несправедлива, поскольку значительно замедляет выполнение трех менее интенсивных тестов, чем libquantum, из-за очень высокой частоты обращений к libquantum в буфере строк (98,4%) и интенсивного использования памяти. Такая несправедливость приводит к самой низкой пропускной способности системы, поскольку ядра, на которых работают три менее интенсивных программы, продвигаются очень медленно. FCFS улучшает справедливость по сравнению с FR-FCFS, потому что предотвращает постоянный приоритет запросов на попадание в буфер строки в libquantum над запросами других потоков. Тем не менее, FCFS по-прежнему несправедливо отдает приоритет интенсивному использованию памяти libquantum и mcf, так как их запросы с большей вероятностью будут старше, чем запросы других потоков. Поскольку характеристики справедливости и пропускной способности как FR-FCFS, так и FCFS были подробно проанализированы в предыдущих исследованиях [25], мы концентрируем наш анализ прежде всего на других алгоритмах планирования.

• NFQ немного улучшает справедливость по сравнению с FCFS, хотя чрезмерно замедляет mcf (в 3,15 раза). У mcf очень высокий уровень параллелизма банков, когда он работает один. Политика NFQ по планированию заключается в балансировке запросов из разных потоков в каждом банке без какой-либо координации между банками. Поскольку другие потоки имеют пакетные шаблоны доступа в некоторых банках, NFQ отдает приоритет их запросам над запросами mcf в этих банках во время пакетов (это частично связано с проблемой простоя, присущей дизайну NFQ, как описано в [25, 31]). Поэтому NFQ разрушает параллелизм банков mcf: в некоторых банках запросы mcf не блокируются запросами из других потоков, в то время как в других банках запросы от пакетных потоков имеют приоритет. Запросы Mcf в этих банках задерживаются, хотя они могли обслуживаться параллельно с другими его запросами. Мы обнаружили, что BLP mcf, равный 4,75, при автономном запуске уменьшается до 2,05 только с NFQ, а его среднее время задержки на доступ к DRAM (AST / req) увеличивается с 64 до 193 тактов процессора.

• STFM обеспечивает лучшую справедливость и пропускную способность, чем все предыдущие политики. Тем не менее, он также значительно снижает (замедляет) mcf, в 2.77X. Это связано с двумя причинами. Во-первых, STFM пытается обеспечить справедливость путем оценки замедления памяти каждого потока и определения приоритетов запросов от потоков, которые наиболее замедлены. STFM наказывает mcf, потому что его эвристика для оценки присущего mcf банковского параллелизма не всегда точна [25] и, следовательно, недооценивает замедление mcf. Во-вторых, как и NFQ, STFM не поддерживает параллелизм: он не пытается обслуживать запросы из потока параллельно. Вместо этого он расставляет приоритеты запросов от потоков, которые, по его оценкам, имели самые высокие запоминания - в этом случае libquantum и GemsFDTD. Запросы этих потоков часто имеют приоритет над запросами mcf, увеличивая AST / req для mcf с 64 до 174 циклов.

• PAR-BS обеспечивает лучшую справедливость и пропускную способность системы. Это снижает несправедливость с 1,42 (STFM) до 1,07 и улучшает взвешенное ускорение на 4,4% и ускорение hmean на 8,4% по сравнению с STFM. Компонент запроса пакета в PAR-BS справедливо распределяет замедления памяти, эффективно сдерживая влияние libquantum на другие потоки. Мы обнаружили, что пакетирование запросов является более эффективным и надежным в обеспечении справедливости, чем методы NFQ и STFM, поскольку оно не уязвимо к 1) проблемам бездействия и баланса доступа к банку в подходе NFQ [25], 2) неверной оценке замедлений потоков в подход STFM. Планирование с учетом параллелизма в пакете позволяет PAR-BS лучше использовать параллелизм банка mcf, поддерживая его AST / req на 146 циклах, ниже, чем NFQ и STFM. Следовательно, PARBS замедляет mcf (на 2,17X) меньше, чем NFQ (3,15X) и STFM (2,77X).

* + 1. **Пример II: не интенсивная нагрузка**

На рисунке 6 показана несправедливость и пропускная способность рабочей нагрузки, включая три неинтенсивных теста и один интенсивный тест. Только одно приложение (omnetpp) имеет высокий параллелизм банков (3.78), что приводит к тому, что среднее время простоя на DRAM-доступ составляет 86 циклов, когда omnetpp запускается один.

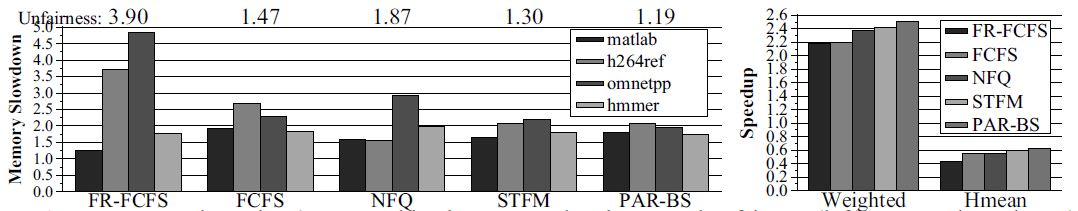


Рисунок 6. 4-ядерная рабочая нагрузка без использования памяти: замедление работы памяти и несправедливость (слева), пропускная способность системы (справа)

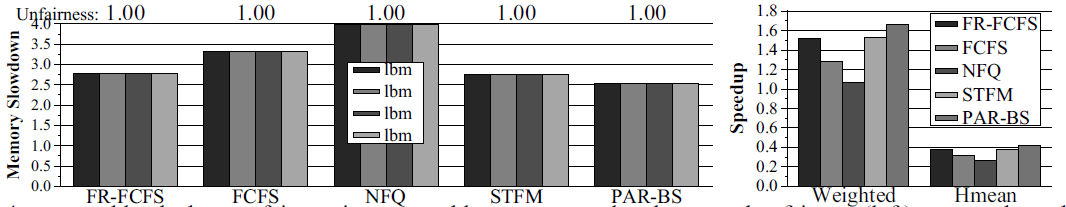


Рисунок 7. 4-ядерная рабочая нагрузка, где несправедливость не является проблемой: замедление памяти и несправедливость (слева), пропускная способность системы (справа)

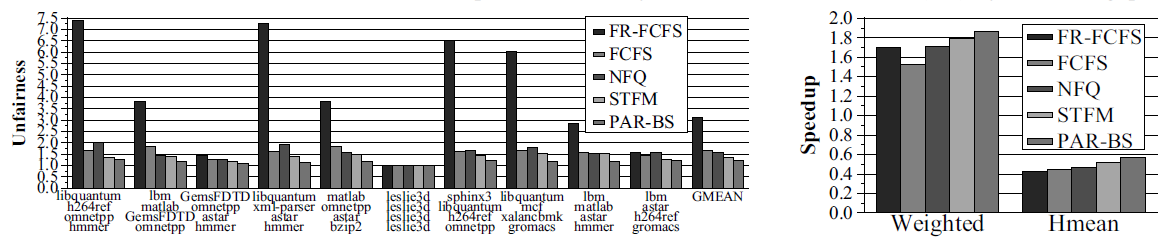


Рисунок 8. Усредненная несправедливость (слева) и пропускная способность системы (справа) (по среднему геометрическому значению) для всех 100 рабочих нагрузок, выполняемых в 4-ядерной системе

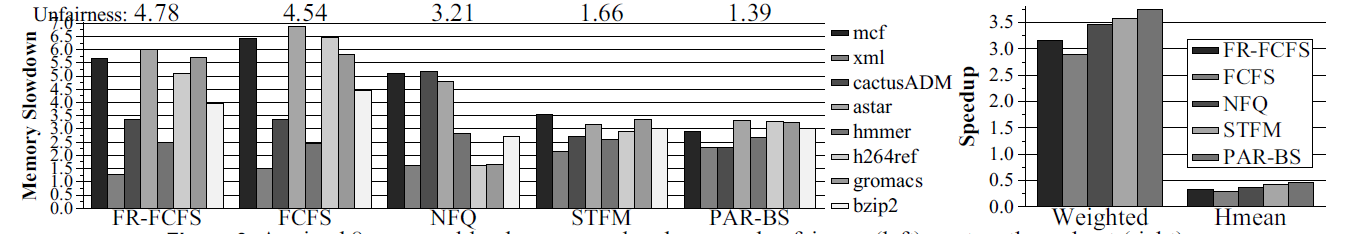


Рисунок 9. Смешанная 8-ядерная рабочая нагрузка: замедление памяти и несправедливость (слева), пропускная способность системы (справа)

PAR-BS является единственным планировщиком, который существенно не наказывает поток с высоким параллелизмом банков (omnetpp). NFQ и STFM уменьшают несправедливость по сравнению с FR-FCFS, поскольку они успешно устраняют проблемы, вызванные жесткой политикой FR-FCFS в отношении попадания строки. Однако ни NFQ, ни STFM не могут возместить потери omnetpp в банковском параллелизме, и оба больше всего замедляют этот поток. На самом деле, NFQ еще более несправедливо, чем FCFS, потому что его схема «ранний виртуальный-крайний срок-первый» отдает приоритет пакетным запросам h264ref (и в меньшей степени также hmmer) над запросами omnetpp в банках, к которым они одновременно обращаются [25]. Это приводит к тому, что обращения к omnetpp, которые в противном случае выполнялись бы параллельно, выходили из синхронизации и становились сериализованными, что снижает производительность omnetpp. Процессор останавливается для задержки доступа к банку каждого доступа, а не амортизирует эту задержку, перекрывая задержки нескольких ожидающих обращений. Результатом является AST / req 256 циклов для omnetpp. Хотя STFM сокращает эту меру до 182 циклов, она все равно чрезмерно замедляет omnetpp, поскольку не в состоянии оптимизировать параллелизм банка omnetpp и недооценивает замедление этого потока. Напротив, PAR-BS с поддержкой параллелизма снижает AST / req для omnetpp до 150 циклов.

PAR-BS превосходит все существующие схемы, достигая наилучшей справедливости, а также улучшая взвешенное и hmean-ускорение на 3,1% и 5,2% по сравнению с STFM, соответственно. В отличие от других схем, это поток с наименьшим объемом памяти (h264ref), который больше всего замедляется PAR-BS, но замедление этого потока тем не менее меньше, чем с другими планировщиками. Некоторые из менее частых запросов h264ref могут пропустить формирование пакета, и в этом случае они не будут обслуживаться, пока пакет не завершится. Однако это не приводит к значительному замедлению, потому что 1) партии быстро обрабатываются из-за небольшой Marking-Cap 5; мы обнаружили, что средняя партия завершается за 1269 циклов, 2) даже если запросы h264ref не помечены, они по-прежнему обслуживаются, если нет отмеченного запроса для требуемого банка, 3) поскольку запросы h264ref редки, они имеют приоритет в пределах пакет из-за нашей Max-Total схемы ранжирования потоков; таким образом, даже если запрос пропускает пакет, он будет обработан первым в следующей партии.

* + 1. **Ситуационное исследование III: эталонный тест с интенсивным использованием памяти с высоким параллелизмом банков, выполняемый с копиями**

Наш последний пример предназначен для демонстрации параллельного поведения планировщика PAR-BS. Для этого мы минимизируем дисперсию между потоками и запускаем четыре одинаковые копии lbm вместе на CMP. Как и ожидалось, в этом случае все планировщики совершенно справедливы (рис. 7 (слева)), но они значительно различаются по замедлению памяти и, следовательно, по пропускной способности системы. FCFS значительно замедляет каждую копию lbm по сравнению с FR-FCFS, потому что он явно не использует локальность буфера строки. Производительность NFQ еще хуже, потому что он не только ограничивает локальность буфера строк, который может использоваться контроллером памяти (используя оптимизацию инверсии приоритетов в [28]), но также часто чередует запросы от различных копий банка в банк, чтобы поддерживать баланс виртуального срока каждой копии lbm. Это разрушает частоту попадания в буфер строк каждой копии lbm, уменьшая ее с 61% до всего лишь 31%, и, следовательно, снижает пропускную способность системы на 29,7%. STFM обеспечивает ту же пропускную способность, что и FR-FCFS, потому что никогда не переключается на политику планирования, ориентированную на справедливость, поскольку правильно оценивает несправедливость в системе как 1.

PAR-BS достигает наилучшей пропускной способности системы, обслуживая параллельные запросы каждой lbm параллельно, уменьшая среднее время простоя, которое DRAM-доступ накладывает на поток (с 222 (FR-FCFS и STFM) и 322 (NFQ) до 199 циклов). Следовательно, PAR-BS улучшает как взвешенное, так и среднее ускорение на 8,6%. Следовательно, обеспечение параллелизма планировщика DRAM повышает пропускную способность системы даже в однородном сочетании приложений, где несправедливость не является проблемой.

* + 1. **4-ядерные эксперименты: средние результаты**

На рисунке 8 (слева) сравнивается несправедливость пяти планировщиков по 10 другим разным рабочим нагрузкам, а также усредняется по всем 100 проверенным рабочим нагрузкам. На рисунке 8 (справа) показана средняя пропускная способность системы на 100 рабочих нагрузок. PAR-BS обеспечивает лучшую справедливость и лучшую пропускную способность. Несправедливость снижена с 1,36 (STFM) до 1,22. В то же время пропускная способность системы повышается на 4,4% (взвешенное ускорение) и на 8,3% (ускорение hmean) по сравнению с лучшей ранее предложенной схемой планирования (STFM).

* 1. **PAR-BS на 8-ядерных и 16-ядерных системах**

Система DRAM станет более узким местом QoS и производительности по мере увеличения количества ядер, разделяющих ее. Мы кратко рассмотрим эффективность PAR-BS на 8-ядерных и 16-ядерных системах. На рисунке 9 показана 8-ядерная рабочая нагрузка, состоящая из 3 приложений, интенсивно использующих память, и 5 приложений, не требующих интенсивного использования. Mcf - единственная программа с очень высоким внутренним параллелизмом. Все предыдущие планировщики последовательно замедляют mcf (по крайней мере, в 3,5 раза), потому что они не могут управлять сериализацией одновременных обращений DRAM от mcf из-за помех от других семи . С другой стороны, PAR-BS сохраняет параллелизм банка mcf, снижение замедления до 2,8Х (а также AST / req с 330 (NFQ) и 221 (STFM) до всего 173 циклов). В результате PAR-BS обеспечивает лучшую справедливость и пропускную способность системы.

На рисунке 10 представлены результаты о несправедливости и пропускной способности в 16-ядерной системе для пяти типовых рабочих нагрузок, а также усредненные по всем 12 рабочим нагрузкам. PAR-BS снижает несправедливость с 1,81 (STFM) до 1,63, одновременно улучшая ускорение по весу на 3,2% и ускорение hmean на 5,1% по сравнению с STFM.

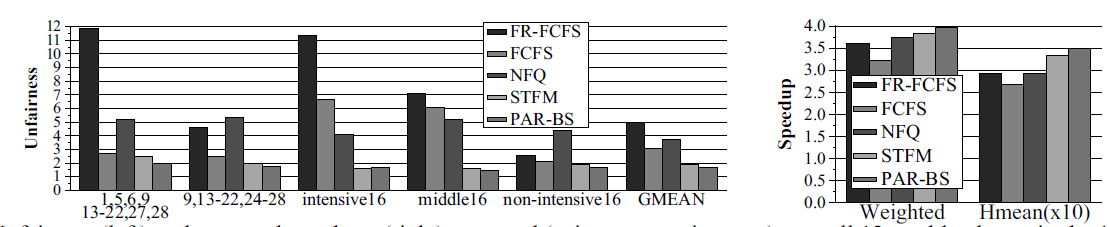


Рисунок 10. Усредненная несправедливость (слева) и пропускная способность системы (справа) (по среднему геометрическому значению) для всех 12 рабочих нагрузок, запущенных в 16-ядерной системе

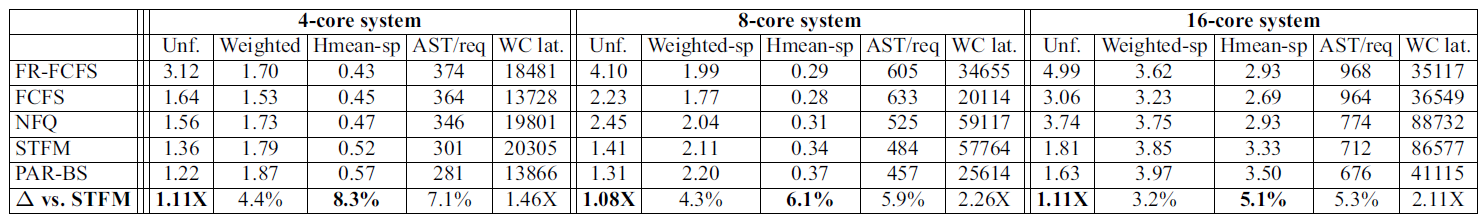


Таблица 4. STFM против других: несправедливость (Unf.), Пропускная способность (weighted / hmean-speedup), AST / req и задержка запроса в худшем случае (WC lat.), Усредненная по всем рабочим нагрузкам

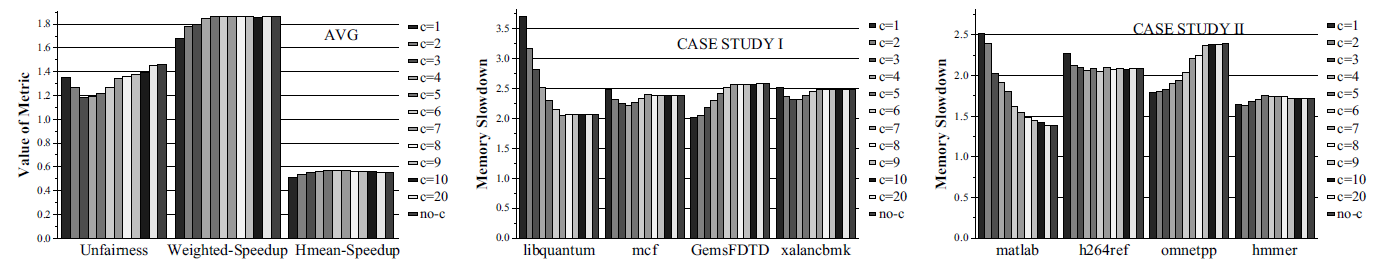


Рисунок 11. Влияние Marking-Cap на несправедливость и пропускную способность (слева); Замедление для Case Study I (в центре) и Case Study II (справа)

**Резюме:** Таблица 4 суммирует нашу оценку, сравнивая среднее геометрическое несправедливости и пропускную способность системы PAR-BS с предыдущими схемами. PAR-BS обеспечивает самое низкое среднее время задержки на запрос, что указывает на то, что оно эффективно снижает среднюю стоимость запроса DRAM на производительность. Кроме того, PAR-BS обеспечивает значительно меньшую задержку запроса в худшем случае, чем другие технологии, учитывающие QoS. Мы обнаружили, что как NFQ, так и STFM могут задерживать запросы от определенных потоков на очень долгое время для обеспечения . Напротив, пакетный компонент PAR-BS достигает справедливости, ограничивая количество времени, в течение которого запросы потока могут задерживаться. PAR-BS последовательно обеспечивает лучшую справедливость и пропускную способность, чем лучший предыдущий метод (STFM) для всех исследованных систем. Мы пришли к выводу, что PAR-BS очень эффективна в обеспечении наилучшей справедливости и высочайшей производительности системы в 4-, 8- и 16-ядерных системах.

* 1. **Анализ**
     1. **Эффект Marking-Cap**

Marking-Cap определяет продолжительность пакета, изменяя количество запросов, отмеченных при формировании нового пакета. Изменение этого параметра влияет на свойства справедливости и пропускной способности PAR-BS, поскольку он изменяет 1) количество используемой локализации буфера строки, 2) количество задержек без опознавательных запросов и 3) степень параллелизма на уровне банка, который можно использовать.

На рисунке 11 (слева) показано влияние изменения Marking-Cap от 1 до 20 и вообще не использования Marking-Cap (no-c) на несправедливость и пропускную способность, усредненную по 100 рабочим нагрузкам в 4-ядерной системе. Когда Marking-Cap наименьший, пропускная способность системы самая низкая, потому что получающиеся партии слишком малы. Например, при ограничении 1 поток может иметь не более 1 запроса на банк в пакете. Такой небольшой размер пакета значительно снижает способность нашего планировщика: 1) использовать локальность буфера строк и 2) находить параллельные обращения к потокам с высоким параллелизмом банков. Если в банке поток A имеет 5 невыполненных запросов в одну строку, а поток B имеет 5 запросов в другую строку, ограничение 1 приводит к чередованию запросов потока A и B, поскольку только один запрос к банку может завершиться с каждого потока в пакете. Это чередование приводит к конфликту строк для каждого доступа и, следовательно, значительно увеличивает задержку каждого потока. В отличие от этого, с маркировочным пределом 5, PAR-BS будет обслуживать сначала 5 запросов А, а затем 5 запросов В со всеми доступами, за исключением того, что первый из каждого потока является попаданием строки. Небольшой ограничитель также приводит к плохой справедливости, потому что он штрафует потоки с высокой локализацией буфера строки (например, libquantum и matlab на рисунке 11 (в центре) и (слева)).

По мере увеличения Marking-Cap несправедливость уменьшается, а пропускная способность системы возрастает до определенного момента, после которого несправедливость увеличивается по двум причинам. Во-первых, большая крышка (cap) позволяет интенсивным потокам в памяти вставлять больше запросов в пакет и, таким образом, задерживает неинтенсивные потоки, чьи запросы чаще «пропускают» формирование пакета. Таким образом, большой предел наказывает менее интенсивно использующие память потоки, как показано в замедлении памяти для GemsFDTD и xalancbmk на рисунке 11 (в центре) и для omnetpp и hmmer на рисунке 11 (справа). Во-вторых, поскольку PAR-BS устанавливает приоритеты для потоков с высокой локализацией буфера строки в пакете, большое ограничение увеличивает задержку потоков с низкой локальностью буфера строки в пакете.

В соответствии с рисунком 11 (слева), маркировочный предел 5 обеспечивает наилучшую среднюю пропускную способность системы (как взвешенное ускорение, так и hmean-speedup), обеспечивая при этом очень хорошую справедливость. Поэтому в наших экспериментах мы используем Marking-Cap 5. Обратите внимание, что можно улучшить наш механизм, сделав Marking-Cap адаптивным.

* + 1. **Эффект пакетного выбора**

На рисунке 12 (слева) сравнивается несправедливость и пропускная способность статического пакетирования с различными вариантами выбора для BatchDuration (варьируются от 400 до 25600 циклов), пакетного сбора и полного пакетирования, которые используются в PAR-BS, которые были описаны в разделе 4.4. На рисунках 12 (в центре) и (справа) показано влияние выбора пакетной обработки на замедление памяти потоков в двух тематических исследованиях. В среднем, полная дозировка обеспечивает лучшую справедливость и пропускную способность.

*Статическое пакетирование* является несправедливым, если BatchDuration слишком мало (например, 400 или 800 циклов). Поскольку большинство запросов в буфере запросов помечается небольшим значением BatchDuration, схема отдает приоритет потокам, интенсивно использующим память, с высокой частотой обращений к буферу строк. Следовательно, небольшая BatchDuration эффективно исключает пакетную обработку запросов и вырождается в политику определения приоритетов по приоритету «сначала попадание строки, ранжирование в первую очередь, сначала старые», которая (аналогично FR-FCFS) наказывает менее интенсивные потоки с низкой локализацией буфера строки, как показано на рисунке 12 (в центре) и (справа). И наоборот, если BatchDuration слишком велико, большинство запросов в буфере не отмечены. Это также эффективно исключает пакетирование запросов и ведет себя подобно FR-FCFS. Преимущество статического пакетирования - BatchDuration в 3200 циклов, но это не обеспечивает такую же хорошую производительность или справедливость, как полное пакетирование, поскольку оно жесткое / неадаптивное и подвержено голоданию.

*Пакетная обработка Eslot* снижает вероятность наказания неинтенсивных потоков. К сожалению, как показано на рисунке 12 (в центре) и (справа), он слишком сильно штрафует потоки, интенсивно использующие память, допуская запросы из менее интенсивных потоков в текущий пакет, что снижает частоту обращений к буферу строк для интенсивных потоков. Хотя в некоторых случаях это может привести к улучшению пропускной способности системы (например, для тематического исследования II на рисунке 12 (справа) - не показано на рисунке), полная группировка обеспечивает лучшую среднюю справедливость и пропускную способность системы. Мы заключаем, что полная пакетная обработка является наиболее эффективной политикой пакетной обработки для PAR-BS.

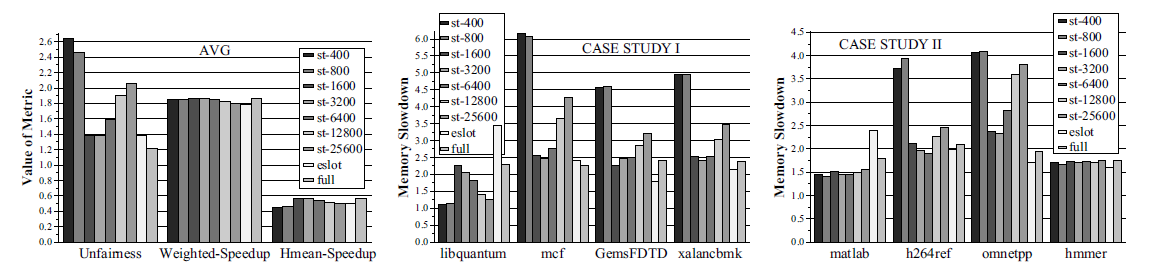


Рисунок 12. Влияние выбора дозирования на несправедливость и пропускную способность (слева); Замедление для Case Study I (в центре) и Case Study II (справа)

* + 1. **Влияние понимания параллельности и различных схем внутрипартийного планирования**

На рисунке 13 (слева) рассматривается влияние изменения схемы ранжирования внутри пакета или ее полного удаления и простого использования FR-FCFS или FCFS для определения приоритетов между командами в пакете. Кроме того, мы изучаем три альтернативные схемы ранжирования внутри пакета, две из которых не придерживаются принципа «самое короткое задание - сначала»: схема случайного ранжирования присваивает потокам случайные ранги при формировании пакета; схема циклического перебора чередует ранг каждого потока в циклическом порядке в последовательных партиях.

На рисунке 13 (слева) показаны эти альтернативные методы внутри пакетного планирования, не основанные на самом коротком задании, значительно ухудшающие справедливость и пропускную способность системы, поскольку они увеличивают среднее время завершения потоков. В частности, изменение схемы ранжирования с Max-Total или Total-Max (которые работают аналогично) на случайную или круговую схему ранжирования снижает взвешенное ускорение/ hmean-speedup соответственно на 5,7% и 9,8%. Отсутствие ранжирования (т. Е. FR-FCFS или FCFS) в пакете полностью исключает понимание параллелизма в нашем предложении, сохраняя при этом компонент пакетирования запросов без изменений. Результатом является снижение как справедливости, так и пропускной способности. Использование политики FR-FCFS в пакете приводит к потере взвешенной скорости / hmean-speedup 4,7% и 10,7% по сравнению с PAR-BS. Как и ожидалось, FCFS обеспечивает лучшую справедливость, чем FR-FCFS, но значительно хуже пропускную способность.

Мы заключаем, что понимание параллелизма является ключевым компонентом нашего предложения. Однако даже без учета параллелизма сама концепция пакетирования запросов приводит к проектам, которые практически конкурируют с лучшим ранее предложенным планировщиком, STFM. Как показано на Рисунке 13 (слева), ранговое ранжирование в пакете обеспечивает чуть худшую справедливость и только на 2,1% / 1,5% меньшую средневзвешенную скорость / hmean-speedup, чем STFM.

На рисунке 13 (в середине и справа) показано, что улучшение пропускной способности за счет приоритетов с учетом параллелизма является значительным, когда потоки имеют высокий собственный параллелизм на уровне банка (4 копии lbm), но незначительным, когда потоки имеют низкий параллелизм (4 копии matlab) , Мы пришли к выводу, что компонент осведомленности о параллелизме нашего предложения не зависит от компонента справедливости и может использоваться для повышения исключительно пропускной способности системы, даже если справедливость не является проблемой.

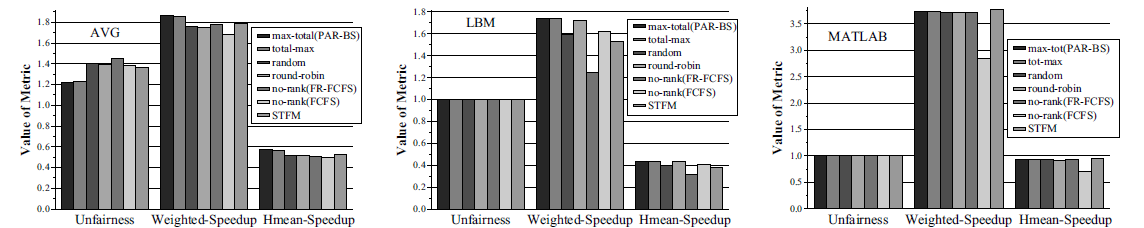


Рисунок 13. Влияние политики внутрипартийного планирования на несправедливость и пропускную способность (слева); замедление для 4 копий lbm (в центре) и matlab (справа)

* 1. **Оценка поддержки приоритетов потоков**

Мы оценили поддержку PAR-BS для приоритетов потоков в различных сценариях и представили два репрезентативных тематических исследования, чтобы подчеркнуть их эффективность. На рисунке 14 (слева) показаны замедления памяти для программ на 4 lbm в минуту с различным весом (для NFQ и STFM) и соответствующими приоритетами (для PAR-BS). Две программы имеют приоритет 1 (соответствует весу 8 в NFQ / STFM), а две имеют приоритеты 2 и 8. Хотя все три планировщика уважают относительные приоритеты потоков, PAR-BS гораздо более эффективна: это приводит к самому низкому замедлению для программ с наивысшим приоритетом, поскольку оно сохраняет их параллелизм в банках. Lbm с приоритетом 1 испытывает замедление на 2,09 и 2,15 с NFQ и STFM, но только на 1,88 с PAR-BS. Кроме того, мы обнаружили, что PAR-BS обеспечивает более высокую пропускную способность системы даже для программ с низким приоритетом (например, lbm с самым низким приоритетом имеет гораздо меньшее замедление с PAR-BS, чем с другими схемами).

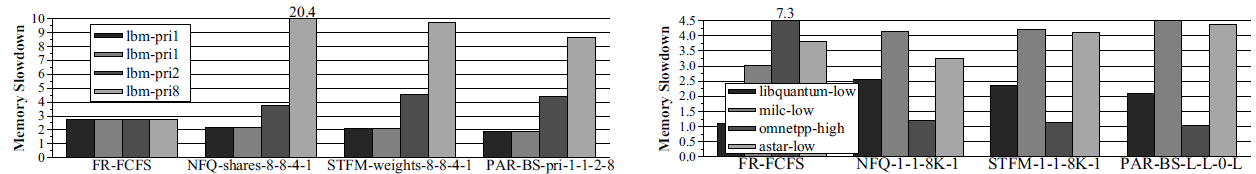


Рисунок 14. Оценка PAR-BS против STFM и NFQ с различными приоритетами / весами потоков

На рисунке 14 (справа) представлен сценарий, в котором omnetpp является наиболее важным потоком для пользователя, тогда как остальные три запланированных потока не важны. Поэтому системное программное обеспечение определяет другие потоки как «оппортунистические», то есть они должны обслуживаться только при наличии доступной полосы пропускания. Как объяснено в разделе 5, PAR-BS легко приспосабливает это понятие «оппортунистического обслуживания», никогда не включая запросы этих потоков в пакет. Для NFQ и STFM понятия «оппортунистический сервис» не существует, поэтому мы аппроксимировали его, назначив очень большой вес (8192) для высокоприоритетного omnetpp и очень малые веса (1) для потоков с низким . BS обеспечивает намного более высокую пропускную способность для высокоприоритетного потока. Замедление работы Omnetpp составляет только 1,04 для PAR-BS, тогда как для STFM оно составляет 1,14, а для NFQ - 1,19. Следовательно, из обоих примеров мы заключаем, что PAR-BS обрабатывает приложения с более высоким приоритетом лучше, чем альтернативные подходы для обеспечения приоритетов / весов потоков в контроллере DRAM.

1. **Сопутствующая работа**

**Справедливые и качественные сервисные контроллеры DRAM**: в последние два года все больше внимания уделяется разработке контроллеров DRAM, ориентированных на обеспечение качества обслуживания в системах с общей памятью. Мы уже предоставили обширные качественные и количественные сравнения двух совсем недавно предложенных контроллеров DRAM, целью которых является обеспечение QoS, планировщика на основе справедливой очереди (NFQ) Nesbit и др. [28] и Mutlu и Moscibroda планировщика для постоянной памяти (STFM) [25]. Рафик и соавт. [31] предложили усовершенствовать схему NFQ, используя справедливую организацию очередей во время начала, которая обеспечивает лучшую справедливость, чем виртуальная справедливая организация очередей во время окончания. Как объяснено в [25], хотя справедливая организация очередей является хорошей абстракцией справедливости для сетевых проводов без состояния какого-либо параллелизма (т. е. банков), она не применима напрямую к DRAM-системам, поскольку она не учитывает состояние буфера строк и параллелизм банков , два критических детерминанта производительности DRAM. Для сравнения, наша конструкция обеспечивает не только справедливость, QoS и свободу голодания, но также значительно повышает пропускную способность системы за счет лучшего перекрытия доступа DRAM внутри потоков.

Iyer et al. [11] делает набросок схемы, которая позволяет запросам только потоков с более высоким приоритетом обходить другие запросы в контроллере памяти. Однако их решение не обеспечивает справедливости для потоков с равным приоритетом. Некоторые контроллеры DRAM [19, 16] достигают жестких гарантий в реальном времени за счет снижения пропускной способности и гибкости, что недопустимо в высокопроизводительных системах общего назначения.

**Пакетирование:** общая концепция «пакетирования» использовалась в планировании диска [7, 39, 12] для предотвращения истощения запросов ввода / вывода. Мы применяем аналогичную концепцию, пакетирования запросов, в нашей конструкции PAR-BS и оцениваем компромиссы, связанные с пакетированием в контроллерах DRAM. Однако компромиссы между локальностью, пропускной способностью и параллелизмом в памяти DRAM очень отличаются от компромиссов в дисках с последовательным доступом, поскольку дисковые накопители не имеют 1) структуры с накоплением или 2) буферов строк.

**Распознавание параллелизма**: Концепция понимания параллелизма на уровне памяти использовалась в процессорных кэшах для улучшения политики замены кеша [30]. Авторы заметили, что ошибки в кеше, которые могут обслуживаться параллельно с другими ошибками, обходятся дешевле по производительности процессора, чем ошибки, возникающие изолированно. Они предложили политику замены, которая пытается сохранить дорогостоящие блоки в кэше. Наше предложение ортогонально: оно активно пытается повысить вероятность того, что пропуски кеша из заданного потока будут обслуживаться параллельно и могут 1) использоваться вместе с и 2) повысить эффективность замены кеша с учетом MLP.

**Оптимизация пропускной способности DRAM**: Zuravleff и Robinson [44] предложили FR-FCFS-подобный планировщик, который стремится максимизировать пропускную способность DRAM. В ряде работ рассматривалось влияние различных политик контроллера памяти и оптимизации пропускной способности DRAM в многопроцессорных / многопоточных [27, 43] и однопоточных системах [33, 20, 32, 10, 34]. Эти методы не учитывают честность или внутри поточный параллелизм банка.

**Справедливость в ресурсах на кристалле**. Предлагаемые методы для справедливого совместного использования кэшей CMP (например, [38, 14]) и ресурсов многопоточного процессора (например, [37, 18, 8]) дополняют нашу работу и могут использоваться совместно с PAR-BS.

1. **Выводы**

Мы представили новое комплексное решение как для высокопроизводительного, так и для QoS-дизайна планировщика DRAM. По сравнению с существующими планировщиками DRAM наш пакетный планировщик с поддержкой параллелизма (PAR-BS) значительно повышает как справедливость, так и пропускную способность системы в системах, где DRAM является общим ресурсом для нескольких потоков. Наша методика объединяет две ортогональные идеи: 1) она обеспечивает справедливость потока и лучше предотвращает краткосрочное и долгосрочное голодание благодаря использованию пакетной обработки запроса; 2) внутри пакета она явно сокращает среднее время остановки потока с помощью параллелизма. Политика планирования DRAM, которая улучшает внутри потоковый параллелизм на уровне банка, используя принцип первоочередного планирования самого короткого задания. Эффективно улучшая справедливость и производительность системы, PAR-BS также легко настраивается. Наша будущая работа будет сосредоточена на формальном анализе свойств параллельности, локальности и справедливости PAR-BS для дальнейшего совершенствования эвристики приоритизации используемых запросов.

**Благодарности**

Мы благодарим Тора Аамодта, Курта Экели, Брэда Бекмана, Тома Конте, Кэти Кунс, Жозе Жоао, Дэйва Харпера, Хайсона Кима, Чанга Джу Ли, Бертона Смита и Кушагру Вейда за их комментарии к предыдущим черновикам этой статьи.