ГЛАВА 5. ОРГАНИЗАЦИЯ КОГЕРЕНТНОСТИ МНОГОУРОВНЕВОЙ ИЕРАРХИЧЕСКОЙ ПАМЯТИ

5.1. Классификация архитектур вычислительных систем

5.1.1. Проблема когерентности памяти ВС

Многопроцессорную ВС можно рассматривать как совокупность процессоров, подсоединенных к многоуровневой иерархической памяти. При таком представлении ВС — коммуникационная среда, объединяющая процессоры и блоки памяти, составляет неотъемлемую часть иерархической памяти. Структурно-технические параметры коммуникационной среды определяют характеристики многоуровневой памяти.

В многопроцессорной ВС для каждого элемента данных должна быть обеспечена когерентность (согласованность, одинаковость) его копий, обрабатываемых разными процессорами и размещенных в разных блоках иерархической памяти. Механизмы реализации когерентности могут быть как явными, так и неявными для прикладного программиста. Используя механизм реализации когерентности в качестве основания, архитектуры ВС можно классифицировать по способу размещения данных в иерархической памяти и способу доступа к этим данным.

5.1.2. Архитектура с явным размещением данных и явным указанием доступа к данным

В этой архитектуре, называемой также архитектурой с распределенной памятью, программист явно задает действия по поддержке когерентности памяти посредством передачи данных, программируемой с использованием специальных команд «послать» (send) и «принять» (receive). Каждый

процессор имеет свое собственное адресное пространство (память ВС распределена), а согласованность элементов данных выполняется путем установления соответствия между областью памяти, предназначенной для передачи командой send, и областью памяти, предназначенной для приема данных командой receive, в другом блоке памяти. Архитектура ВС с распределенной памятью требует от программиста синхронизации обращений к разным блокам памяти и явного указания на необходимые пересылки данных. Распределение данных по блокам и уровням иерархической памяти также реализуется программистом.

5.1.3. Архитектура с неявным размещением данных и неявным указанием доступа к данным

В этой архитектуре, называемой также архитектурой с разделяемой памятью, механизм реализации когерентности прозрачен для прикладного программиста, и в программах отсутствуют какие-либо другие команды обращения к памяти, кроме команд чтения (load) и записи (store). Используется единое физическое или виртуальное адресное пространство. Архитектура ВС с разделяемой памятью имеет много привлекательных черт:

- однородность адресного пространства памяти, позволяющая при создании приложений не учитывать временные соотношения между обращениями к разным блокам иерархической памяти;
- создание приложений в привычных программных средах;
- легкое масштабирование приложений для исполнения на разном числе процессоров и разных ресурсах памяти.

5.1.4. Архитектура с неявным размещением данных и явным указанием доступа к данным

В этой архитектуре используется разделяемое множество страниц памяти, которые размещаются на внешних устройствах. При явном запросе страницы автоматически обеспечивается когерентность путем пересылки уже запрошенных ранее страниц не из внешней памяти, а из памяти модулей, имеющих эти страницы.

5.1.5. Архитектура с явным размещением данных и неявным указанием доступа к данным

Фирма Encore Computer Corporation запатентовала технологию MEMORY CHANNEL эффективной организации кластерных систем на базе модели разделяемой памяти. Суть технологии заключается в следующем. В каждом компьютере кластера предполагается организация памяти на основе механизма виртуальной адресации. Адрес при этом состоит из двух частей: группы битов, служащих для определения номера страницы, и собственно адреса внутри страницы. В каждом компьютере в ходе инициализации выделяется предписанное, возможно разное, вплоть до полного отсутствия, количество физических страниц памяти, разделяемых этим компьютером с другими компьютерами кластера.

После установления во всех компьютерах отображения страниц памяти доступ к удаленным памятям выполняется посредством обычных команд чтения (load) и записи (store), как к обычным страницам виртуальной памяти, без обращений к операционной системе или библиотекам времени исполнения.

Межкомпьютерные передачи происходят при выполнении команд чтения или записи, адресующих специфичную страницу адресного пространства шины модуля, в которой расположены таблицы отображения разделяемых страниц.

Каждый компьютер кластера имеет встраиваемую в него интерфейсную платуадаптер «шина компьютера входной и выходной каналы (линки) некоторой среды передачи данных». В области адресов устройств ввода/вывода шины размещаются две таблицы управления страницами памяти, соответственно одна для выдачи обращений в удаленные разделяемые (общие) страницы памяти других компьютеров, в другая — для приема обращений из других компьютеров в локальные разделяемые страницы рассматриваемого компьютера.

Каждый элемент таблицы, используемый при выдаче обращений, содержит:

1) данные, необходимые для доставки сообщения в другой компьютер кластера (например, ID-идентификатор компьютера, в памяти которого находится разделяемая страница);
2) данные, необходимые для точного указания места в странице, к которому должен быть осуществлен доступ по чтению или записи;
3) служебные данные, указывающие, например, на состоятельность рассматри ваемого элемента, особенности маршрутизации и т. д.

На основе этих данных адаптер формирует сообщения (пакеты), которые передаются через выходной линк в сеть передачи данных. Будучи доставленным по этой сети в компьютер-адресат, сообщение воспринимается через входной линк адаптером этого компьютера. Сообщение содержит либо команду чтения в совокупности с адресом блока данных, который необходимо прочитать и передать в компьютер, выполняющий команду чтения, либо сообщение содержит команду записи в совокупности с адресом, указывающим на место записи данных, и сами записываемые данные. Возможна также передача в сообщениях сигналов прерываний для удаленной шины и синхронизирующих примитивов, необходимых для взаимного

исключения одновременного доступа совместно протекающих процессов к областям разделяемой памяти. Фактически, операции чтения и записи прозрачно переносятся с шины одного компьютера на шину другого, в котором расположены данные, адресуемые в операции.

5.2. Механизмы неявной реализации когерентности

5.2.1. Аппаратно-программные реализации механизмов когерентности

Современные микропроцессоры имеют один или несколько уровней внутри кристальной кэш-памяти. Поэтому интерфейс микропроцессоров с необходимостью включает механизм организации когерентности внутрикристальной кэш-памяти (например, первого L1 и второго L2 уровней) и внекристальной памяти. Внекристальная память может также быть многоуровневой: состоять из кэш-памяти (например, третьего L3 уровня) и основной памяти.

Реализация механизма когерентности в ВС с разделяемой памятью требует аппаратурно-временных затрат. Причем уменьшить временную составляющую затрат можно за счет увеличения аппаратурной составляющей и наоборот. Уменьшение временной составляющей требует создания специализированной аппаратуры реализации когерентности. Уменьшение аппаратурной составляющей предусматривает некоторый минимум аппаратных средств, на которых осуществляется программная реализация механизма когерентности.

5.2.2. Однопроцессорный подход

Для каждого уровня интеграции микроэлектроники можно говорить об определенном объеме блока основной памяти со временем доступа, близким

к времени доступа к микросхемам памяти. С возрастанием объема блока памяти выше указанного возрастает время доступа.

Идеальная основная память должна иметь «неограниченно» наращиваемый объем и «малое» время доступа к данным. Время обращения к памяти по произвольным адресам должно быть одинаковым. Прямолинейное построение из микросхем памяти блока сколь угодно большого объема со свойствами идеальной основной памяти невозможно. Однако, в связи с тем, что характеристики памяти определяются не только физическими характеристиками микросхем памяти, но и режимом ее использования, прибегают к архитектурно-структурным приемам, учитывающим специфику обращений к памяти пользовательских программ и позволяющим создать при работе этих программ эффект малого времени доступа к большой основной памяти.

Однако в рамках однопроцессорных систем достижимый объем основной памяти с временем доступа, характерным для микросхем памяти, ограничен. Для создания памяти больших объемов прибегают к построению многопроцессорных систем. Кэш-память имеет совокупность строк (cache-line), каждая из которых состоит из фиксированного количества адресуемых единиц памяти (байтов, слов) с последовательными адресами. Типичный размер строки: 16, 32, 64, 128, 256 байтов.

В многоуровневой иерархической памяти эффект уменьшения времени доступа в память достигается за счет динамической локализуемости обрабатываемых данных в кэш-памяти и их многократного использования. Однако это дает эффект только на задачах, допускающих такую локализуемость. Если не удается упорядочить и локализовать адреса доступа в память, производительность падает до скорости доступа в оперативную память и даже ниже, так как кэш-память работает в режиме: загрузка строки

кэш-памяти (32 байта и более), однократное-двукратное использование 4-8 байтов, выгрузка строки кэш-памяти и переход к загрузке очередной строки.

Наиболее часто используются три способа организации кэш-памяти, отличающиеся объемом аппаратуры, требуемой для их реализации. Это так называемые кэш-память с прямым отображением (direct-mapped cache), наборно ассоциативная кэш-память (set-associative cache), и ассоциативная кэш-память (fully associative cache).

При отсутствии необходимой строки в кэш-памяти, одна из ее строк должна быть заменена на требуемую. Используются разнообразные алгоритмы определения заменяемой строки, например циклический, замена наиболее редко используемой строки, замена строки, к которой дольше всего не было обращений, и другие.

Соответствие между данными в основной памяти и кэш-памяти обеспечивается внесением изменений в те области основной памяти, для которых данные в кэш-памяти подверглись модификации. Эти внесения изменений данных выполняются параллельно с вычислениями. Существует несколько способов реализации внесения изменений (и, соответственно, несколько режимов работы кэш-памяти).

Один способ предполагает внесение изменений в основную память сразу после их возникновения в кэш-памяти. При этом процессор простаивает в ожидании завершения записи в основную память. В основной памяти поддерживается правильная копия данных кэш-памяти, и при замене строк не требуется никаких дополнительных действий. Кэш-память, работающая в таком режиме, называется памятью со сквозной записью (write-through).

Другой способ предполагает отображение изменений в основной памяти только в момент вытеснения строки данных из кэш-памяти. Если

адрес памяти, в который необходимо произвести запись, находится в кэш-памяти, то идет запись только в кэш-память. При отсутствии адреса в кэш-памяти производится запись в основную память. Такой режим работы кэш-памяти получил название обратная запись (write-back). Кэш-память с обратной записью создает меньшую нагрузку на шину процессора и обеспечивает большую производительность, однако контроллер для write-back кэш-памяти значительно сложнее, чем у кэш-памяти со сквозной записью.

Существуют также промежуточные варианты (buffed write-through), при которых запросы на изменение в основной памяти буферизуются и не задерживают процессор на время производства записи в память. Эта запись выполняется по мере возможности доступа контроллера кэш-памяти к основной памяти.

Реализация механизма когерентности чаще всего осуществляется с использованием отслеживания (snooping) запросов на шине, связывающей процессор, память и интерфейс ввода/вывода. Контроллер кэш-памяти отслеживает адреса памяти, выдаваемые процессором, и если адрес соответствует данным, содержащимся в одной из строк кэш-памяти, то отмечается «попадание в кэш», и данные из кэш-памяти на правляются в процессор. Если данных в кэш-памяти не оказывается, то фиксируется «промах», и инициируются действия по доставке в кэш-память из основной памяти требуемой строки. В ряде процессоров, выполняющих одновременно совокупность команд, допускается несколько промахов прежде, чем будет запущен механизм замены строк.

Возникает проблема с когерентностью данных в кэш-памяти и областях памяти, используемых при вводе выводе внешними устройствами. При вводе данных необходимо отслеживать на шине обновление данных в

строках основной памяти, копии которых находятся в кэш-памяти, и вести одновременно запись данных в строку кэш-памяти и строку основной памяти.

При выводе данных на внешнее устройство также необходимо отслеживать на шине выдачи данных с соблюдением когерентности данных основной памяти и кэш-памяти.

5.2.3. Многопроцессорный подход

Построение параллельных многопроцессорных систем служит подходом к реализации большой разделяемой (общедоступной) основной памяти, сочетающим использование идей многоуровневой иерархической памяти и параллельного доступа к данным. Под разделяемой основной памятью (shared memory) понимается память с единым адресным пространством, доступ к которой выполняется командами чтения (load) и записи (store).

Основная проблема при создании разделяемой памяти в многопроцессорных системах состоит в том, как извещать другие процессоры об изменениях в разделяемой памяти, вызванных выполнением команд записи в каждом из процессоров. Характерной тенденцией, обусловленной стремлением к повышению производительности микропроцессоров, служит переход от внутрикристальных кэш-памятей со сквозной записью к кэш-памятям с буферизированной записью и к кэш-памятям с обратной записью. При буферизации и обратной записи возрастает промежуток времени между изменением данных в локальной копии данных, размещенной в кэш-памяти микропроцессора, и их появлением на шине памяти микропроцессора. Следовательно, доставка этих изменений в удаленные копии тех же данных в других блоках памяти, включая кэш-память других микропроцессоров, также будет происходить с задержкой.

Кроме того, при обратной записи модификации данных в к памяти принципиально не наблюдаемы на шине памяти, кроме моментов смены кэш-строк.

Для модификации удаленных копий данных используются коммуникационные протоколы и протоколы согласованности (когерентности) состояния памяти. Коммуникационный протокол применяется для доставки изменений. Протокол когерентности должен предотвращать использование копий данных, подвергшихся модификации в другом процессоре. В простейшем случае, например при объединении процессоров и памятей шиной, осуществляется прослушивание (отслеживание) шины, имеющее целью обнаружить выполнение команд записи в адреса ячеек памяти, копии которых размещены в кэш-памятях и основной памяти. В этом случае коммуникационный протокол и протокол когерентности неразрывно связаны.

Для сохранения когерентности данных в соответствии с интуитивно понятной моделью строгой состоятельности памяти операция чтения из разделяемой памяти должна возвращать последнее записанное в читаемую ячейку значение. Поэтому, когда одна из множества копий модифицируется, остальные копии должны быть объявлены несостоятельными (стать устаревшими) либо модифицироваться. Выполнение этих действий, с одной стороны, требует аппаратных средств быстрого распространения изменений, а с другой стороны, какими бы быстрыми ни были эти действия, следование модели строгой состоятельности ведет к снижению производительности. Последнее обусловлено обеспечением непременной синхронности обращений процессоров к памяти и введением глобального времени, позволяющего упорядочить действия, выполняемые во всех процессорах системы. Поэтому для повышения производительности необходимо применение моделей ослабленной состоятельности памяти, допускающих

появление несогласованности копий данных в ходе их параллельной обработки с последующим обеспечением когерентности копий так, чтобы результат исполнения программы был таким же, как при строгой по стоятельности.

Конкретная реализация разделяемой памяти характеризуется:

- уровнем реализации (аппаратный, программный, аппаратно-программный);
- структурой разделяемых данных (слово, кэш-строка, страница, сегмент, фрагмент, объект и т. д.);
- моделью состоятельности памяти, определяющей допустимые последовательности доступа в память, генерируемые пользовательскими программами (SRSW — один читатель/один писатель, MRSW — много читателей/один писатель, MRMW — много читателей/много писателей);
- политикой реализации когерентности данных (модификация, объявление несостоятельными);
- механизмом управления распределением памяти и размещением данных (централизованный/распределенный, статический/динамический). В настоящее время используется ряд моделей состоятельности памяти;
- строгая (strict) — каждая операция чтения возвращает последнее записанное значение;
- последовательная (sequential) — все процессоры в системе наблюдают один и тот же порядок выполнения операций записи и чтения, выполняемых в любом процессоре системы (процессор, выполняющий запись, приостанавливается до получения подтверждений об

объявлении несостоятельными всех копий модифицируемых данных или о модификации этих копий);

- процессорная (processor) — наблюдаемый в двух процессорах порядок выполнения операций чтения/записи может быть не совпадающим, но порядок выполнения записей, производимых каждым процессором, должен быть одним и тем же;
- слабая (weak) — вводящая разграничение между обычным и синхронизованным доступами в память с выполнением требования состоятельности памяти только для доступов в точках синхронизации и разрешением данным быть несогласованными вне этих точек;
- свободная (release) — уточненная модель слабой состоятельности, требующая состоятельности памяти только между парой операций синхронизации ***acquire-release*** (***acquire*** открывает критический интервал и обеспечивает исключительный доступ процессора к разделяемым данным, ***release*** закрывает критический интервал, разрешая всем процессорам доступ к разделяемым данным); при этом операции синхронизации, выполняемые в разных процессорах, должны удовлетворять модели последовательной или процессорной состоятельности памяти;
- неторопливо-свободная (lazy release) — модель свободной состоятельности с отложенным до момента непосредственной востребованности распространением модификаций разделяемой памяти, так что при операции ***acquire*** передаются все предшествующие результаты команд записи, которые выполнялись до наступившего момента синхронизации;

- нетерпеливо-свободная (eager release) — модель свободной состоятельности с передачей изменений разделяемой памяти при реализации операции ***release***, безотносительно к тому, когда эти изменения будут востребованы;
- интервальная свободная (scope) — модель свободной состоятельности с разбиением исполнения на глобальные и локальные интервалы и гарантированием состоятельности разделяемых данных для всех процессоров только в конце каждого глобального интервала, а также гарантированием состоятельности разделяемых данных для последовательных локальных интервалов внутри каждого процессора;
- последовательно-свободная (entry) — вариант модели свободной состоятельности, который требует доступа к разделяемым данным, защищенного синхронизирующими переменными, значения которых должны модифицироваться согласно модели последовательной согласованности, то есть все процессоры должны иметь одну и ту же последовательность обращений к каждой из этих переменных.

В современных микропроцессорах, используемых для построения мультипроцессорных систем, идентичность данных в кэш-памяти ВМ (когерентность кэш-памяти) поддерживается с помощью межмодульных пересылок. Существует несколько способов реализации когерентности, применяемых в зависимости от типа используемой коммуникационной среды и сосредоточенности или физической распределенности памяти между процессорными модулями.

5.3. Аппаратный уровень реализации разделяемой памяти

5.3.1. Архитектуры систем с разделяемой памятью

При аппаратном уровне реализации разделяемой памяти объем разделяемых единиц данных, как правило, составляет кэш-строку. Это обусловлено привязкой к механизму когерентности внутрикристальной многоуровневой кэш-памяти микропроцессоров. Вследствие этого аппаратные реализации универсальны в том смысле, что они не требуют введения в программы дополнительного кода для организации разделяемой памяти. Аппаратные реализации, вследствие применения специально разрабатываемых средств, имеют малую задержку при передаче данных и обеспечивают высокую пропускную способность коммуникационной среды. Однако применение специальных аппаратных средств существенно повышает стоимость таких реализаций по сравнению с программной реализацией разделяемой памяти. При аппаратурном уровне реализации разделяемая память доступна всем программам, как операционной системы, так и прикладным. Все ячейки памяти одинаково доступны, что создает единый системный образ.

Достаточно условно системы с разделяемой (общей) памятью можно разбить на два класса:

- системы с физически одной сосредоточенной общей памятью;
- системы с физически распределенной по вычислительным модулям (ВМ) разделяемой памятью.

Системы, имеющие единую разделяемую память, в основном представлены симметричными мультипроцессорами (SMP).

Системы с распределенной разделяемой памятью подразделяются на:

- системы с архитектурой NUMA (None Uniform Memory Architecture) с неоднородной архитектурой памяти;
- системы с архитектурой COMA (Cache-Only Memory Architecture);
- системы с рефлексивной памятью (RM Reflective Memory).

В архитектурах NUMA, COMA используется объявление данных несостоятельными, а в системах с рефлективной памятью применяется модификация разделяемых данных. В системах SMP возможно как объявление модифицируемых данных несостоятельными, так и модификация разделяемых данных.

5.3.2. Симметричные мультипроцессоры с сосредоточенной памятью

Симметричные мультипроцессоры обычно состоят из небольшого количества процессоров, объединенных с памятью коммуникационной средой (шиной или коммутатором типа «кроссбар»), пропускная способность которой достаточна для поддержания быстрого доступа в память. Время обращения к физически одной общей памяти одинаково для всех процессоров, отсюда произошло одно из названий UMA (Uniform Memory Access) этого класса систем.

Как правило, в SMP применяется протокол когерентности, основанный на «прослушивании» всеми процессорами коммуникационной среды и проведении согласованных изменений кэш-памятей и основной памяти. Модель состоятельности памяти строгая, либо последовательная, либо процессорная.

Рассмотрим реализацию одного из алгоритмов поддержки когерентности кэш-памятей, известного как MESI (Modified, Exclusive, Shared, Invalid). Алгоритм MESI представляет собой организацию когерентности кэш-памяти с обратной записью. Этот алгоритм предотвращает лишние передачи данных между кэш-памятью и основной памятью. Так, если данные в кэш-памяти не изменялись, то незачем их пересылать. Кроме того, возможны еще усовершенствования, которые,

например, применены в микропроцессоре Intel 80860XP, связанные с уменьшением количества сквозных записей.

Для представления алгоритма поддержки когерентности кэш-памятей MESI зададим некоторые начальные условия и введем определения. Итак, каждый ВМ имеет собственную локальную кэш-память, имеется общая разделяемая основная память, все ВМ подсоединены к основной памяти посредством шины. К шине подключены также внешние устройства. Важно понимать, что все действия с использованием транзакций шины, производимые ВМ и внешними устройствами, с копиями строк, как в каждой кэш-памяти, так и в основной памяти, доступны для отслеживания всем ВМ. Это является следствием того, что в каждый момент на шине передает только один, а воспринимают все, подключенные к шине абоненты. Поэтому если для объединения ВМ используется не шина, а другой тип коммутационной среды, то для работоспособности алгоритма MESI необходимо обеспечение вышеуказанного порядка выполнения транзакций.

Каждая строка кэш-памяти ВМ может находиться в одном из следующих состо яний:

M — строка модифицирована (доступна по чтению и записи только в этом ВМ, потому что модифицирована командой записи по сравнению со строкой основной памяти);

E — строка, монопольно копированная (доступна по чтению и записи в этом ВМ и в основной памяти);

S — строка множественно копированная или разделяемая (доступна по чтению и записи в этом ВМ, в основной памяти и в кэш-памятях других ВМ, в которых содержится ее копия);

I — строка, невозможная к использованию (строка не доступна ни по чтению, ни по записи).

Состояние строки используется, во-первых, для определения процессором ВМ возможности локального, без выхода на шину, доступа к данным этой кэш-строки в кэш-памяти, а во-вторых, для управления механизмом когерентности. Для управления режимом работы механизма поддержки когерентности используется бит WT, состояние 1 которого задает режим сквозной (write-through) записи, а состояние 0 — режим обратной (write-back) записи в кэш-память.

При исполнении команд чтения и записи состояние строки кэш-памяти, к которой выполняется доступ, определяется табл. 5.1.

Исходное состояние строки	*Состояние после чтения*	*Состояние после записи*
I	Если WT = 1, тогда E, иначе S; Обновление строки путем ее чтения из основной памяти	Сквозная запись в основную память; I
S	S	Сквозная запись в основную память; Если WT = 1, тогда E, иначе S
E	E	M
M	M	M

Промах чтения в кэш-памяти заставляет вызвать строку из основной памяти и сопоставить ей состояние E или S. Кэш-память заполняется только при промахах чтения. При промахе записи транзакция записи помещается в буфер и посылается в основную память при предоставлении шины.

При несостоятельной строке в состоянии I команда чтения данного из этой строки вызывает чтение строки из основной памяти, размещение ее в кэш-памяти и изменение состояния этой строки в кэш-памяти на E или S. Состояние E будет, если установлен режим сквозной записи, при котором запись производится и в строку кэш-памяти и в строку основной памяти.

Состояние S устанавливается при режиме обратной записи, что позволяет модифицировать данные строки кэш-памяти без немедленной модификации строки основной памяти, что, в свою очередь, увеличивает производительность. До тех пор пока к данным строки не будет доступа других ВМ или внешних устройств, не будет обратной записи, и ВМ не будет использовать шину,

При состоянии I строки команда записи в эту строку изменяет только содержи мое строки основной памяти (сквозная запись), но не изменяет содержимое кэш-памяти и сохраняет состояние строки I.

В состоянии S строки чтение данного из этой строки не меняет ее состояние. Если установлен режим сквозной записи, то после завершения записи состояние строки меняется на E, при режиме обратной записи выполняется сквозная запись, но состояние строки остается прежним S.

Если состояние строки E, то чтение сохраняет это состояние, а запись переводит строку в состояние M. Наконец, если состояние строки M, то как команды чтения, так и команды записи не меняют этого состояния.

Для поддержки когерентности строк кэш-памяти при операциях ввода-вывода и обращениях в основную память других процессоров на шине генерируются специальные циклы опроса состояния кэш-памятей. Эти циклы опрашивают кэш-памяти на предмет хранения в них строки, которой принадлежит адрес, используемый в операции, инициировавшей циклы опроса состояния. Возможен режим принудительного перевода строки в состояние I, который задается сигналом INV. При этом состояние строк определяется табл. 5.2.

Исходное состояние	*INV = 0*	*INV = 1*
I	I	I
S	S	I
E	S	I
M	S; обратная запись строки	I; обратная запись строки

5.3.3. Системы с архитектурой NUMA

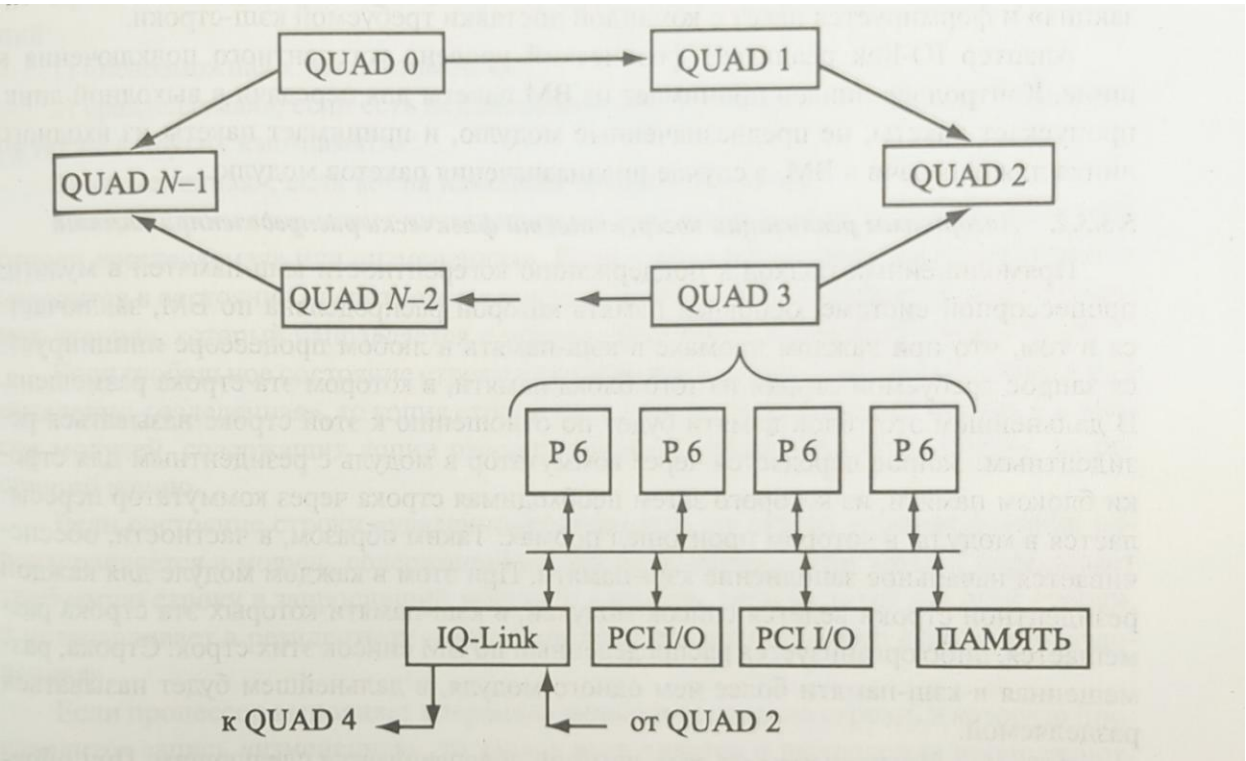
5.3.3.1. Системы с физически распределенной памятью

Физически распределенная разделяемая память (DSM — distributed shared memory) образуется как совокупность памятей ВМ. ВМ могут включать один или несколько процессоров с подсоединенным к ним блоком локальной памяти. В случае использования нескольких процессоров ВМ имеет архитектуру SMP. Распределение памяти по ВМ позволяет достичь рационального использования процессоров и блоков памяти и рациональной компоновки системы, в том числе из коммерчески доступных COTS (Component Off-The-Shelf) вычислительных модулей. В последнем случае в качестве ВМ могут, например, выступать серийные системные платы перс о нальных компьютеров или рабочих станций.

Основная идея архитектуры ccNUMA использование для образования мультипроцессорных систем интерфейса ВМ между кэш-памятью и основной памятью (термин «ccNUMA» употребляется для подчеркивания увязки механизма работы кэш-памяти ВМ с доступом в удаленные блоки памяти других ВМ). Протокол когерентности типа MESI обеспечивает механизм поддержки когерентности кэш-памяти и удаленной основной памяти. Время доступа процессора в свою локальную память в несколько раз меньше, чем время доступа через коммуникационную сеть в память другого ВМ.

Возможно достаточно большое разнообразие вариантов построения таких систем. В качестве примера рассмотрим реализацию NUMA Q2000 Sequent. Структура системы показана на рис. 5.1.

ВС NUMA — построена из модулей SHV (Standard High Volume) Quad, содержащих SMP, включающий до 4 процессоров Pentium Pro. Каждый ВМ помимо 4 процессоров имеет память объемом до 4 Гбайт, 2 моста «шина P6-шина PC» и адаптер 10-link, реализующий протокол когерентности, а также входной и выходной каналы по стандарту SCI.



Для построения ccNUMA в адаптер IQ-link вводится дополнительный блок кэш-памяти уровня L3, называемый удаленной кэш-памятью (Remote Cache), который содержит копии строк данных, загружаемых из блоков памяти других ВМ. Эта кэш-память, с одной стороны, обслуживает обращения к данным при отсутствии требуемой строки во внутрикристальной кэш-памяти и блоке локальной памяти, размещенном на

той же плате. С другой стороны, при отсутствии в этой удаленной кэш-памяти строки, резидентной в блоке памяти другого ВМ, эта строка доставляется в кэш-память в соответствии с протоколом когерентности SCI. Для обозначения архитектуры систем ccNUMA, ВМ которых содержат в своем составе удаленную кэш-память, употребляется также термин NUMA-RC.

В системах с архитектурой NUMA локальные памяти ВМ отображаются в глобальное адресное пространство. Интерфейс адаптера IQ-link с шиной P6 отслеживает адреса, появляющиеся в запросах на шине, и определяет диапазон адресов, к которому принадлежит появившийся адрес. Каждый участок диапазона адресов заранее приписан блоку памяти одного из ВМ. Тем самым определяется идентификатор ВМ. в котором размещены данные, адрес которых появился на шине. Этот идентификатор используется для маршрутизации пакета, доставляющего в случае необходимости требуемую кэш-строку. Для быстрого определения строк, расположенных в L3 кэше SCI, используется специальный каталог. В случае отсутствия в нем тега строки, соответствующей адресу, появившемуся на шине P6, шина переводится в состояние «незавершенная транзакция» и формируется пакет с командой доставки требуемой кэш-строки.

Адаптер IQ-link реализует физический уровень когерентного подключения к шине. Контроллер линков принимает из ВМ пакеты для передачи в выходной линк, пропускает пакеты, не предназначенные модулю, и принимает пакеты из входного линка для передачи в ВМ, в случае предназначения пакетов модулю.

5.3.3.2. Алгоритмы реализации когерентности физически распределенной памяти

Прямолинейный подход к поддержанию когерентности кэш-памятей в мультипроцессорной системе, основная память которой распределена по ВМ, заключается в том, что при каждом промахе в кэш-память в любом процессоре инициируется запрос требуемой строки из того блока памяти, в котором эта строка размещена. В дальнейшем этот блок памяти будет по отношению к этой строке называться резидентным. Запрос передается через коммутатор в модуль с резидентным для строки блоком памяти, из которого затем необходимая строка через коммутатор пересылается в модуль, в котором произошел промах. Таким образом, в частности, обеспечивается начальное заполнение кэш-памяти. При этом в каждом модуле для каждой резидентной строки ведется список модулей, в кэш-памяти которых эта строка размещается, либо организуется распределенный по ВМ список этих строк. Строка, размещенная в кэш-памяти более чем одного модуля, в дальнейшем будет называться разделяемой.

Собственно когерентность кэш-памятей обеспечивается следующим. При обращении к кэш-памяти в ходе операции записи данных, после самой записи, процессор приостанавливается до тех пор, пока не выполнится последовательность действий: измененная строка кэш-памяти пересылается в резидентную память модуля, затем, если строка была разделяемой, она пересылается из резидентной памяти во все модули, указанные в списке разделяющих эту строку. После получения подтверждений, что все копии изменены, резидентный модуль пересылает в процессор, приостановленный после записи, разрешение продолжать вычисления.

Изложенный алгоритм обеспечения когерентности хотя и является логически работоспособным, но практически редко применяется из-за больших простоев процессоров при операциях записи в кэш-строки. На практике применяют более сложные алгоритмы, обеспечивающие меньшие

простои процессоров, например DASH, представление о котором приведено ниже.

Каждый модуль памяти имеет для каждой строки, резидентной в модуле, список модулей, в кэш-памяти которых размещены копии этой кэш-строки. С каждой строкой в резидентном для нее модуле связаны три ее возможных глобальных состояния:

1) «некэшированная», если копия строки не находится в кэш-памяти какого-либо другого модуля, кроме, возможно, резидентного для этой строки;
2) удаленно-разделенная», если копии строки размещены в кэш-памятях других модулей;
3) «удаленно-измененная», если строка изменена операцией записи в каком-либо модуле.

Кроме этого, каждая кэш-строка находится в одном из трех локальных состояний:

1) «невозможная к использованию»;
2) «разделяемая», если есть неизмененная копия, которая, возможно, размещается также в других кэш-памятях;
3) «измененная», если копия изменена операцией записи.

Каждый процессор может читать из своей кэш-памяти, если состояние читаемой строки «разделяемая» или «измененная». Если строка отсутствует в кэш-памяти или находится в состоянии «невозможная к использованию», то посылается запрос — «промах чтения», который направляется в модуль, резидентный для требуемой строки.

Если глобальное состояние строки в резидентном модуле «некэшированная» или «удаленно-разделенная», то копия строки посылается в запросивший модуль и в список модулей, содержащих копии

рассматриваемой строки, вносится модуль запросивший копию. Если состояние строки «удаленно-измененная», то запрос «промах чтения» перенаправляется в модуль, содержащий измененную строку. Этот модуль пересылает требуемую строку в запросивший модуль и в модуль, резидентный для этой строки, и устанавливает в резидентном модуле для этой строки состояние «удаленно-раздленная».

Если процессор выполняет операцию записи и состояние строки, в которую производится запись «измененная», то запись выполняется и вычисления продолжаются. Если состояние строки «невозможная к использованию» или «разделяемая», то модуль посылает в резидентный для строки модуль запрос на захват в исключительное использование этой строки и приостанавливает выполнение записи до получения подтверждений, что все остальные модули, разделяющие с ним рассматриваемую строку, перевели ее копии в состояние «невозможная к использованию».

Если глобальное состояние строки в резидентном модуле «некэшированная», то строка отсылается запросившему модулю, и этот модуль продолжает приостановленные вычисления.

Если глобальное состояние строки «удаленно-разделенная», то резидентный модуль рассылает по списку всем модулям, имеющим копию строки, запрос на переход этих строк в состояние «невозможная к использованию». По получении этого запроса каждый из модулей изменяет состояние своей копии строки на «невозможная к использованию» и посылает подтверждение исполнения в модуль, инициировавший операцию записи. При этом в приостановленном модуле строка после исполнения записи переходит в состояние «удаленно-измененная.

Предпринимаются попытки повысить эффективность реализации алгоритма когерентности, в частности, за счет учета специфики

параллельных программ, в которых используются асинхронно одни и те же данные на каждом временном интервале исключительно одним процессором с последующим переходом обработки к другому процессору. Такого рода ситуации случаются, например, при определении условий окончания итераций. В этом случае возможна более эффективная схема передачи строки из кэш-памяти одного процессора в кэш-память другого процессора.

5.3.3.3. Особенности реализации алгоритма когерентности

В системах, использующих простые (однокаскадные) коммутаторы с временным разделением (шину) или пространственным разделением (crossbar), интерфейс с этим коммутатором каждого модуля «отслеживает» (snoops) все передачи адресов через коммутатор. Каждый модуль проверяет состояние своей локальной кэш-памяти и исполняет необходимые для реализации протокола когерентности действия с кэш-строками.

В системах с составными распределенными коммутаторами необходимо вести списки кэш-строк, что требует дополнительных ресурсов памяти. Когерентность обеспечивается хранением информации о состоянии местонахождении каждой строки общей разделяемой памяти. Элемент списка в каждом процессоре содержит эту служебную информацию для одной строки, а также указатели «вперед» и «назад» по списку других строк.

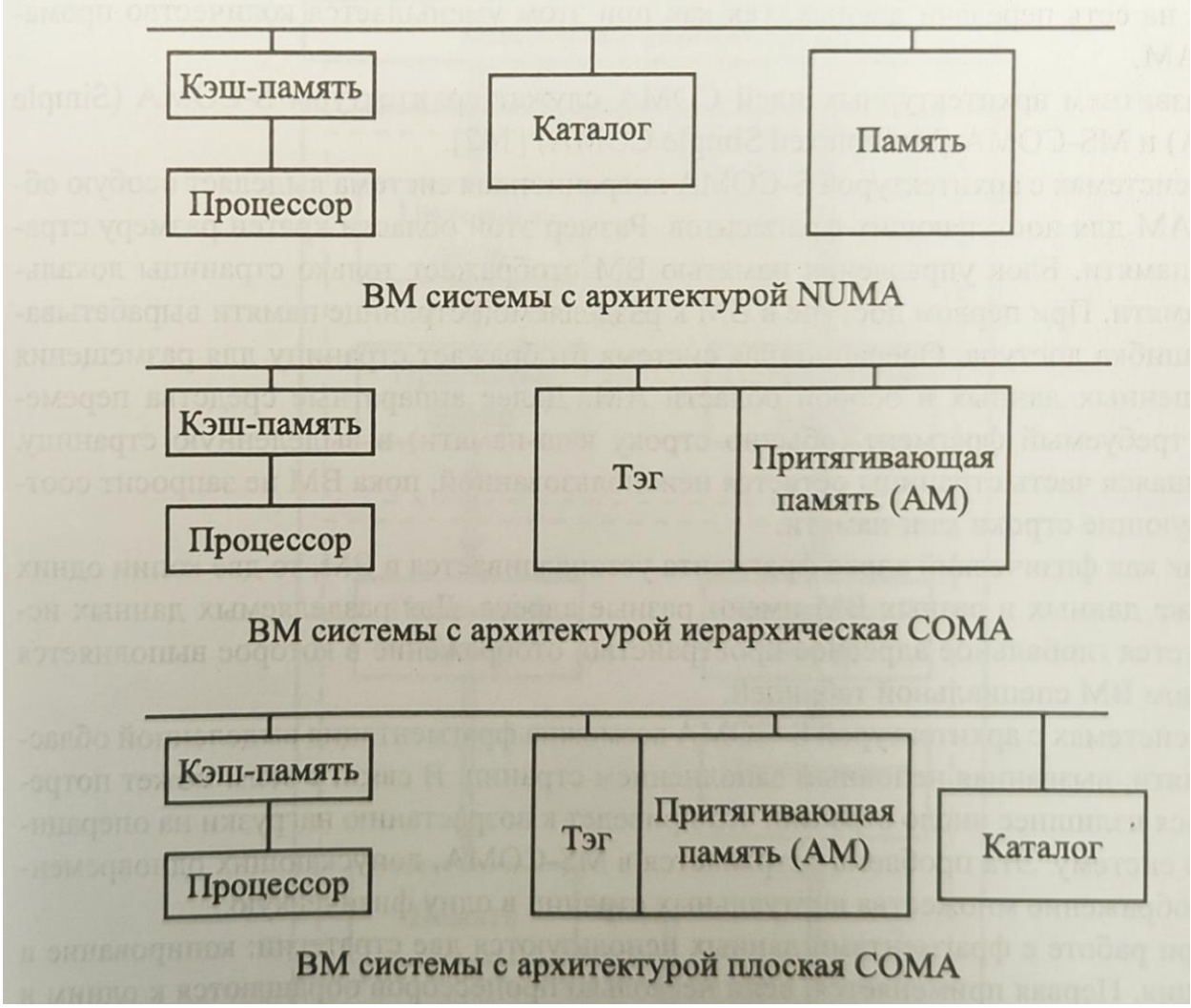
5.3.4. Системы с архитектурой COMA

В системах с архитектурой COMA модули памяти ВМ строятся как кэш-память, которую называют притягивающей памятью (attraction memory-AM). При запросе фрагмента памяти из другого ВМ поступивший фрагмент размещается как в кэш-памяти запрашивающего ВМ, так и в его АМ. Ранее размещенный в АМ фрагмент может быть удален из АМ, если нет места для вновь поступившего другого фрагмента.

АМ содержит для каждого фрагмента данных тег, включающий адрес и состояние фрагмента. При промахе в кэш-память контроллер памяти ВМ просматривает теги АМ для определения возможности извлечения требуемого фрагмента из локальной АМ. При отсутствии фрагмента в локальной АМ вырабатывается запрос на его доставку. Для поиска фрагмента в таких известных системах, как KSR-1 и DDM, используются специальные аппаратные средства сети межмодульных связей.

В KSR-1 сеть строится как древовидная иерархия колец, в которой ВМ служат листовыми элементами. Соответственно в DDM сеть представляет собой древовидную иерархию шин. Эти системы называются также нерархическими COMA. Каждый уровень древовидной структуры включает каталог, содержащий сведения обо всех фрагментах, размещенных в нижележащих ВМ-листьях. Поэтому при поиске фрагмента необходимо просматривать каталоги вышележащих уровней древовидной структуры до тех пор, пока не будет обнаружен требуемый фрагмент.

В системах с так называемой плоской COMA архитектурой (flat-COMA) в каждом ВМ вводится каталог для указания местоположения резидентных в этом ВМ фрагментов. В этих системах может применяться обычная высокоскоростная сеть в отличие от специализированной, используемой в иерархической COMA. При промахе в АМ по адресу требуемого фрагмента определяется резидентный ВМ, в каталоге которого содержится информация о местонахождении требуемого фрагмента. В резидентный ВМ направляется запрос фрагмента. Запрос или вызывает передачу фрагмента, или, при отсутствии в резидентной АМ фрагмента, перенаправляется согласно каталогу к месту нахождения фрагмента для удовлетворения запроса. Соответствующие структуры ВМ с архитектурами NUMA, иерархическая COMA и плоская COMA приведены на рис. 5.2.



Притягивающая память АМ функционирует во многом подобно обычной кэш-памяти. Однако для фрагментов АМ нет резидентного блока памяти, в который помещается модифицированный фрагмент при его удалении из АМ. Поэтому фрагмент, удаляемый из АМ одного ВМ, должен быть помещен в АМ другого ВМ. В связи с тем что возможно одновременное существование нескольких копий одного фрагмента, одна из копий объявляется основной. В отличие от прочих копий, могущих быть просто затертыми, если они не модифицировались, основная копия должна быть обя зательно перемещена в другую АМ.

Перемещение фрагмента в АМ называется впрыскиванием. Возможно впрыскивание как в предварительно выбранный по специальному запросу ВМ, у которого есть место в АМ, так и в произвольно назначаемый ВМ. В последнем случае возможно возникновение цепочки впрыскиваний, в том числе циклической, что требует дополнительных механизмов для разрешения этих ситуаций. Еще одним решением может быть перемещение фрагмента в тот ВМ, из которого он поступил первоначально.

Отношение объема памяти, используемого прикладной программой, к общему суммарному объему АМ всех ВМ называется давлением. Чем меньше давление, тем больше памяти остается для копий фрагментов данных и тем меньше нагрузка на сеть передачи данных, так как при этом уменьшается количество промахов в АМ.

Развитием архитектурных идей COMA служат архитектуры S-COMA (Simple COMA) и MS-COMA (Multiplexed Simple COMA).

В системах с архитектурой S-COMA операционная система выделяет особую область АМ для поступающих фрагментов. Размер этой области кратен размеру страницы памяти. Блок управления памятью ВМ отображает только страницы локальной памяти. При первом доступе в ВМ к разделяемой странице памяти вырабатывается ошибка доступа. Операционная система отображает страницу для размещения запрошенных данных в особой области АМ. Далее аппаратные средства перемещают требуемый фрагмент (обычно строку кэш-памяти) в выделенную страницу. Оставшаяся часть страницы остается неиспользованной, пока ВМ не запросит соответствующие строки кэш-памяти.

Так как физический адрес фрагмента устанавливается в ВМ, то две копии одних и тех же данных в разных ВМ имеют разные адреса. Для

разделяемых данных используется глобальное адресное пространство, отображение в которое выполняется в каждом ВМ специальной таблицей.

В системах с архитектурой S-COMA возможна фрагментация выделенной области памяти, вызванная неполным заполнением страниц. В связи с этим может потребоваться излишнее число страниц, что приведет к возрастанию нагрузки на операционную систему. Эта проблема устраняется в MS-COMA, допускающих одновременное отображение множества виртуальных страниц в одну физическую.

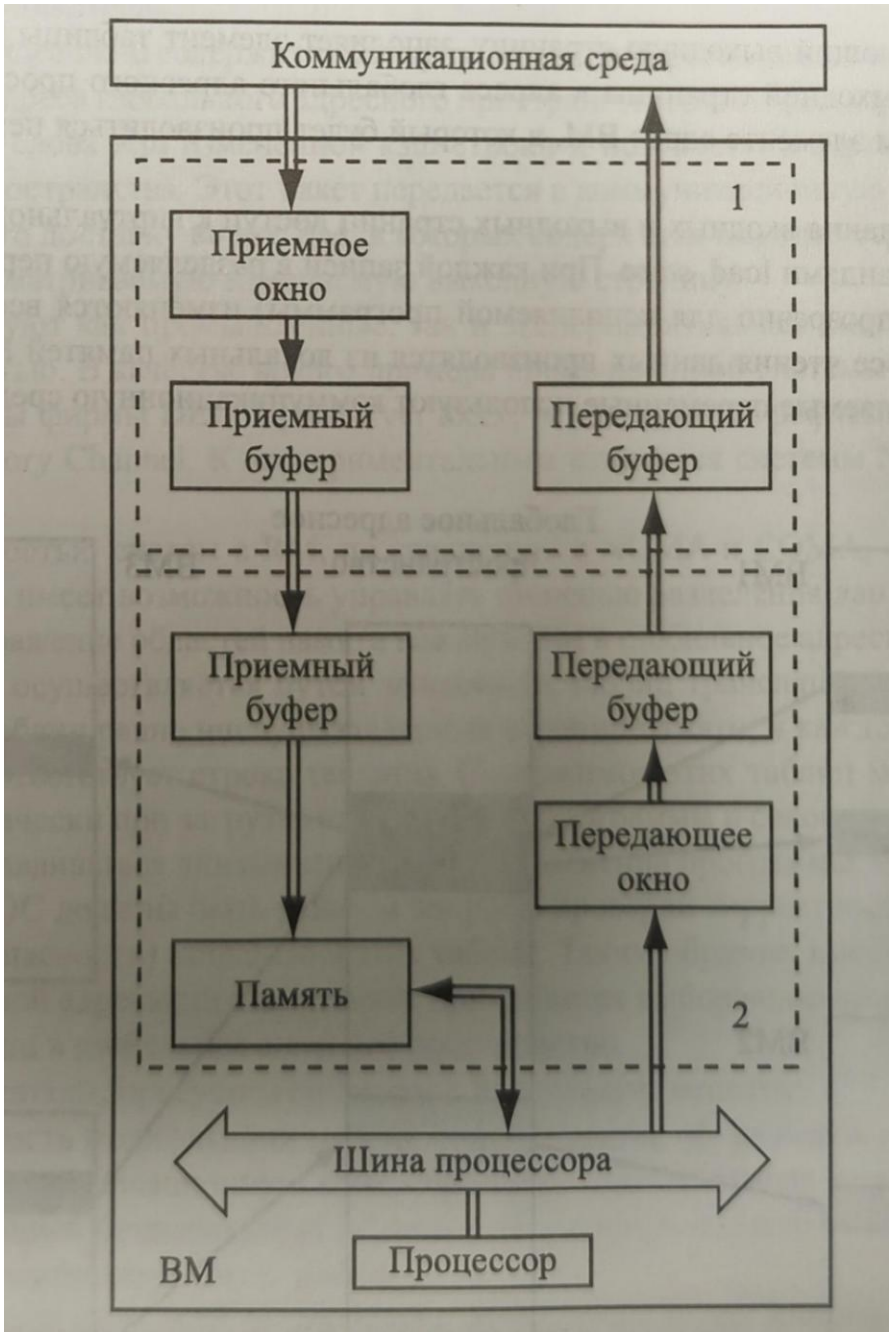
При работе с фрагментами данных используются две стратегии: копирование и миграция. Первая применяется, если несколько процессоров обращаются к одним и тем же данным в основном по чтению, для чего требуется размещение копий в нескольких АМ. Вторая если фрагмент данных модифицируется в каждый момент только одним процессором, что требует размещения фрагмента только в одной АМ.

В зависимости от того, преобладает в прикладной программе копирование или миграция, оказывается более эффективной либо архитектура NUMA-RC, либо Flat-COMA. Сравнение делается в предположении о равенстве объема аппаратных средств: ВМ системы с архитектурой NUMA-RC имеют кэш-память третьего уровня, объем которой равен объему памяти АМ в ВМ сопоставляемой системы с архитектурой Flat-COMA.

5.3.5. Системы с рефлексивной памятью

Основное свойство рефлексивной памяти, обусловившее ее название, состоит в том, что каждая копия разделяемого данного в локальной памяти любого ВМ имеет одно и то же значение.

Функционирование системы с рефлексивной памятью (RM) поясним рис. 5.3. Итак, система состоит из коммуникационной среды и совокупности ВМ. Каждый ВМ имеет процессор, блок памяти, шину процессора, по которой он обращается в память, и адаптер коммуникационной среды (сетевой интерфейс). Адаптер состоит из приемо-передающей части коммуникационной среды (1) и приемо-передающей части процессора (2).



Память гранулирована на страницы. Совокупность страниц локальной памяти ВМ делится на две группы: локальные неразделяемые страницы и глобальные разделяемые страницы. Рефлексивная память образуется из всех этих распределенных по различным блокам физической памяти глобально разделяемых страниц памяти, отображенных в глобальное разделяемое адресное пространство.

ВМ, создающий входную страницу, выделяет страницу физической памяти и делает ее доступной для разделения с другими модулями. В разделяемом глобальном адресном пространстве эта страница получает адрес, который используется для доступа к ней из выходной страницы другого или того же самого ВМ. Указанное отображение адресов глобального адресного пространства в локальные адреса выполняется таблицей преобразования адресов, размещенной в приемном окне адаптера ВМ.

ВМ, создающий выходную страницу, заполняет элемент таблицы преобразования адресов выходной страницы в адреса глобального адресного пространства, задавая в каждом элементе адрес ВМ, в который будет производиться пересылка данных.

После создания входных и выходных страниц доступ к виртуальной памяти выполняется командами load, store. При каждой записи в разделяемую переменную автоматически (прозрачно для исполняемой программы) изменяются все копии этой переменной. Все чтения данных производятся из локальных памятей ВМ, и только записи в разделяемые переменные используют коммуникационную среду, объединяющую ВМ.

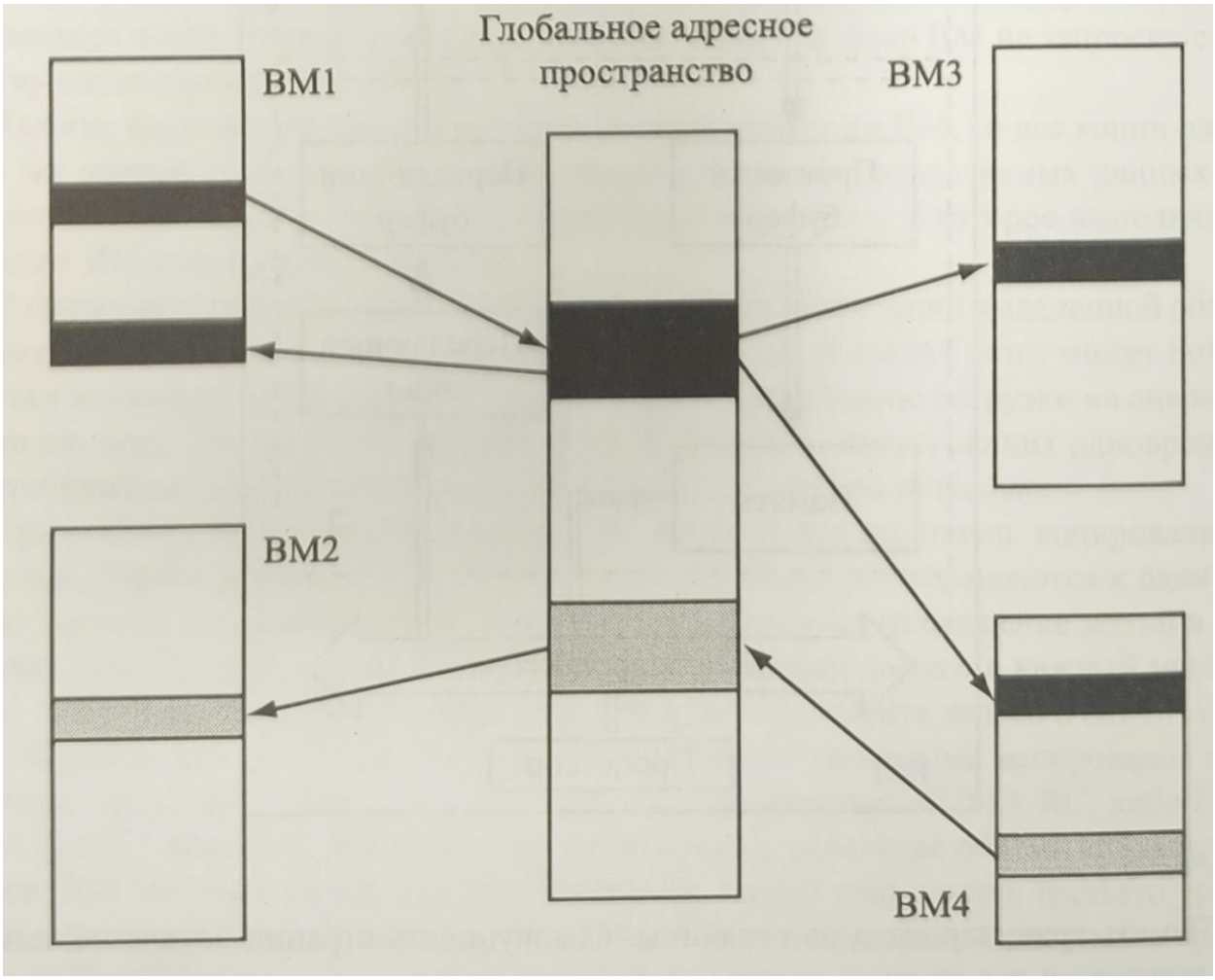


Рис. 5.4 показывает два множества разделяемых страниц. ВМ1 имеет одну выходную страницу, глобальный адрес которой совпадает с глобальными адресами входных страниц в ВМ3, ВМ4 и в самом ВМ1. ВМ4 содержит одну выходную страницу, глобальный адрес которой равен глобальному адресу входной страницы ВМ2.

Если ВМ выполняет команду записи в переменную, расположенную в неразделяемой памяти, то эта команда изменяет содержимое только локальной памяти процессора. Однако команда записи в переменную, принадлежащую разделяемой выходной странице, помимо изменения переменной этой страницы активизирует передающее окно адаптера.

Передающее окно содержит таблицу трансляции адресов разделяемой выходной страницы в адреса глобального адресного пространства. Затем формируется пакет из измененного слова или измененной кэш-строки и полученного адреса глобального адресного пространства. Этот пакет передается в коммуникационную среду, обеспечивающую его доставку во все ВМ, в которых содержатся входные страницы, разделяющие рассматриваемую изменяемую выходную страницу.

Существуют как промышленные, так и экспериментальные системы с рефлексивной памятью. В качестве яркого примера промышленной системы можно упомянуть кластеры фирмы DEC AlphaServer 8xxx, использующие рефлексивную память на базе Memory Channel. К экспериментальным относятся системы Merlin, Sesame, Shrimp.

Особенностью систем с RM, по сравнению с NUMA и COMA, служит то, что пользователь имеет возможность управлять степенью разделения данных в системе, так как отображение областей памяти каждого ВМ в глобальное адресное пространство системы осуществляется путем заполнения таблиц трансляции адресов. Число строк этих таблиц равно числу разделяемых страниц памяти, и каждой странице однозначно соответствует строка таблицы. Содержимое этих таблиц может задаваться либо статически при загрузке параллельной программы в совокупность ВМ, либо может устанавливаться динамически в ходе исполнения программы. В любом случае средствами ОС должны быть решены вопросы проверки корректности (согласованности и безопасности) заполнения этих таблиц. Таким образом, вместо фиксированной глобальной адресации всей памяти применяется выборочное отображение отдельных страниц в глобальное адресное пространство.

К недостаткам, присущим системам с RM, следует отнести:

- сложность модификации многих копий страниц, обусловленную большой на грузкой на коммуникационную среду при широковещательных передачах пакетов или организацией бездедлоковых передач пакетов при последовательном обходе ВМ с копиями модифицируемых страниц;
- излишней нагрузкой на коммуникационную среду при многократной модификации, сопровождаемой передачей пакета, одной и той же глобальной переменной в случае, если необходимо только конечное значение переменной после завершения ее преобразований в этом ВМ;
- невозможность использовать в качестве разделяемых страницы, помещаемые во внутрикристальную кэш-память с обратной записью, так как модификация этих страниц внутри микропроцессора не может быть обнаружена на его шине, что ограничивает организацию RM только на уровне некэшируемых страниц, например, адресного пространства шины PCI;
- существенно более продолжительная длительность модификации всех копий страниц по сравнению с доступом в локальную память ВМ требует явного введения барьерной синхронизации ВМ при записи в глобальные переменные для поддержания когерентности памяти в рамках модели свободной состоятельности.

В системах с рефлексивной памятью могут использоваться различные модели состоятельности памяти от строгой и процессорной до различных вариантов свободной состоятельности.

5.4. Программный уровень реализации разделяемой памяти

5.4.1. Системы на базе передачи сообщений

В настоящее время системы на базе коммерчески доступных микропроцессорных материнских плат и сетевых интерфейсов широко используются как дешевая ап\паратная альтернатива системам с аппаратной реализацией распределенной разделяемой памяти. Протоколы когерентности (согласованности состояния) памяти при программной реализации служат надстройкой над аппаратными средствами передчи сообщений. Это, с одной стороны, заведомо создает проблемы с недостаточной пропускной способностью при создании удаленных копий данных, передаче изменений данных и миграции данных, но, с другой стороны, ничем не ограничивает разнообразие протоколов согласованности состояния памяти, давая возможность учитывать особенности исполняемых прикладных программ.

Строгая когерентность требует, чтобы каждый ВМ видел все доступы в память в одном и том же порядке. Однако обычно в параллельных программах используются операции синхронизации для управления доступом к разделяемым переменным или для установления порядка выполнения операций. В первом случае используются операции синхронизации для установления критических интервалов, во втором барьерная синхронизация. Программа может не учитывать изменения разделяемых переменных вне критического интервала, что позволяет использовать модели свободной состоятельности памяти. Поэтому ослабленная модель состоятельности памяти делает различие между нормальным и синхронизированным доступами в память.

Модели ослабленной состоятельности памяти дают возможность повышения производительности и расширяют количество аппаратных платформ с различными коммуникационными протоколами, на которых возможно программно организовать разделяемую память, быть может, при определенных ограничениях на последовательности команд записи/чтения в

прикладных программах. Реализация этих ограничений возлагается на синхронизирующие примитивы, посредством которых программист формирует допустимые последовательности команд.

В большинстве программных реализаций в качестве наименьших элементов разделяемой памяти служат страницы, что обусловлено использованием существующих аппаратных средств организации виртуальной памяти для обнаружения записей в разделяемые страницы (например, запись в страницу, доступную только для чтения, или несостоятельную страницу). Однако сравнительно большой объем разделяемых страниц создает проблему ложного разделения, при котором конфликтными признаются модификации разными процессорами различных ячеек памяти одной страницы, на самом деле не конфликтующие между собой. Решением служит либо введение логических разделяемых единиц (например, объектов), либо отображение многих виртуальных страниц в одну физическую страницу.

Программный уровень реализации разделяемой памяти может базироваться либо на компиляторе, который выявляет доступы к разделяемым переменным и вставляет в исполняемый код вызовы синхронизирующих примитивов и процедур обеспечения когерентности, либо на библиотеке процедур, статически или динамически связываемых с исполняемой прикладной программой. В любом случае обращение к разделяемым данным, расположенным в другом ВМ, вызывает пересылку данных через коммуникационную среду с обеспечением когерентности всех копий одних и тех же данных в разных ВМ.

Обнаружение модифицированных данных требует либо внесения в код программы дополнительных команд для установки битов модификации для указания измененных данных, либо создания перед первой записью в

страницу копии разделяемых данных, которая используется по завершении интервала вычислений процесса для сравнения текущего состояния данных страницы с их сохраненной копией. Для установлення моментов модификации разделяемых данных вводятся отметки логического времени, привязывающие изменения к интервалам синхронизированного доступа к разделяемым данным.

Эффективность программных реализаций разделяемой памяти ограничивается большим размером разделяемых единиц (страниц), временем выполнения операций синхронизации, включая определение модифицированных данных, а также задержкой и пропускной способностью коммуникационной среды. В программных реализациях DSM задержка доступа в удаленную память на порядок и более превосходит задержку в аппаратных реализациях DSM. Поэтому сравнимая производительность может быть достигнута лишь на некоторых классах задач.

При программном уровне реализации разделяемая память доступна только ограниченному классу прикладных программ, написанных с использованием ограниченного круга языковых конструкций, ориентированных на поддержку выявления компилятором записей в разделяемые переменные или использование библиотек для доступа к этим переменным. Не все ячейки памяти одинаково доступны. Разделяемая память вследствие этого не может использоваться для реализации синхронизации. Для синхронизации используется ограниченное количество синхронизирующих примитивов, например критические интервалы (lock) и барьеры (barriers), реализованные посредством передачи сообщений. Стремление к повышению эффек тивности ведет к применению моделей состоятельности памяти, значительно отличающихся от используемых в системах с аппаратной поддержкой разделения памяти. Программы,

созданные для систем с апаратной разделяемой памятью, трудно переносимы на системы с программной разделяемой памятью.

5.4.2. Реализация модели свободной состоятельности памяти

Для реализации модели свободной состоятельности (Release Consistency) могут быть использованы различные протоколы, в том числе неторопливой свободной состоятельности (Lazy Release Consistency LRC) и нетерпеливой (Eager Release consistency ERC) свободной состоятельности памяти.

В рамках этих моделей задержки, вызванные передачей модифицируемых данных, скрываются за счет буферизации записей или их конвейеризации в пределах критических интервалов. При этом в критическом интервале возможно объединение групп записей в одну, что повышает эффективность их передачи через коммуникационную среду.

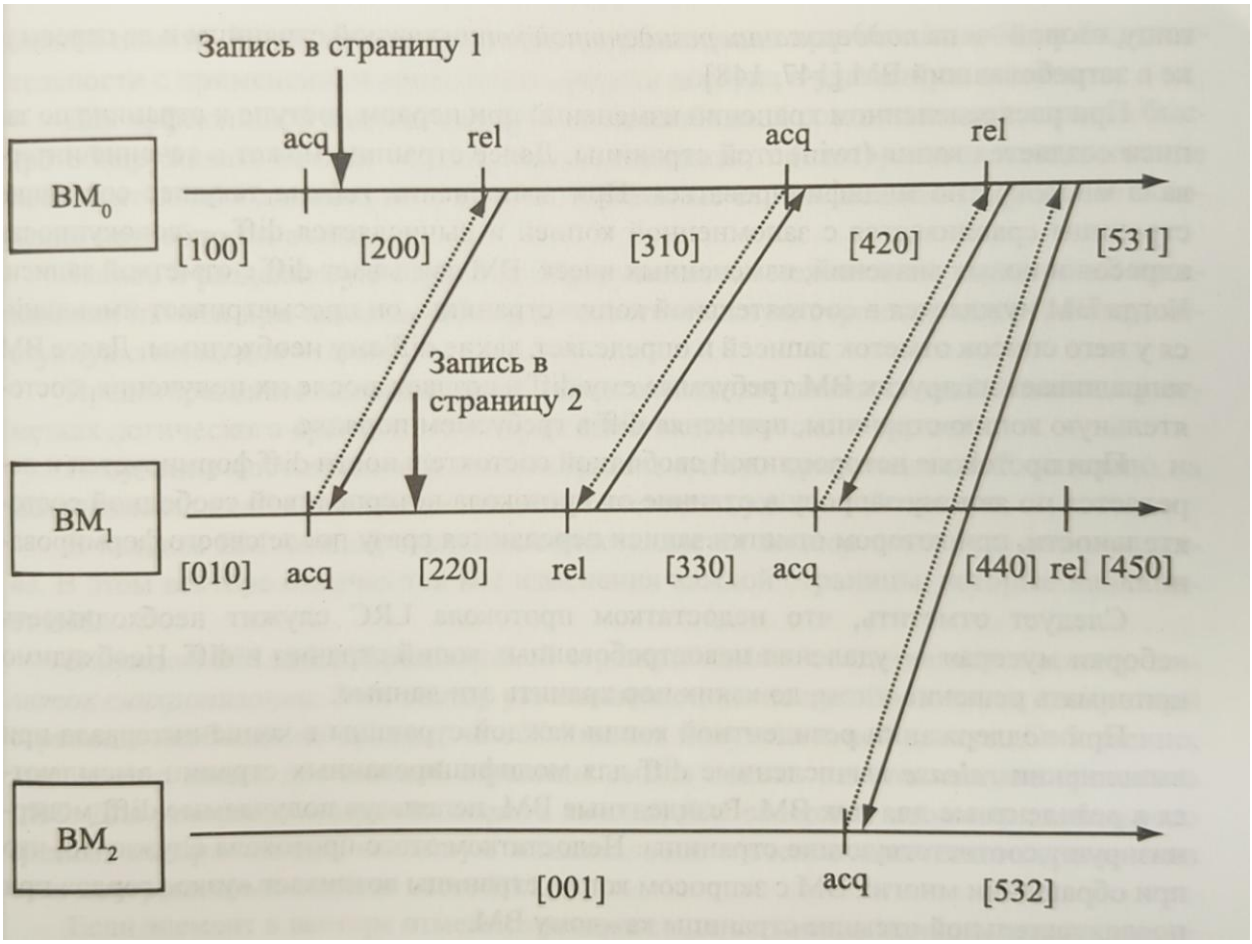
Все ВМ выполняют обычные операции чтения/записи и операции синхронизации ***acquire*** и ***release***. По условиям модели должна быть обеспечена последовательная состоятельность выполнения операций синхронизации, то есть все ВМ должны видеть один и тот же порядок выполнения этих операций. Одним из механизмов реализации установления частичной упорядоченности операций синхронизации служит введение в каждом ВМ локальной отметки времени и вектора отметок времени. При свободной состоятельности время протекания процесса в каждом ВМ де лится на интервалы. После выполнения каждой операции ***acquire*** (s) или ***release*** (s) начинается новый интервал, и локальная отметка времени ВМ увеличивается на 1. Вектор отметок времени служит для поддержки временных зависимостей между всеми ВМ. Этот вектор имеет компоненты, одназначно сопоставленные каждому ВМ.

Каждая модификация памяти описывается отметкой записи (write-notice). Когда ВМ пишет в разделяемую страницу, он создает отметку записи для этой страницы. В эту отметку помещаются значение локальной отметки времени интервала, адрес и значение изменяемого слова. Каждый ВМ формирует список отметок записи для каждой страницы.

5.4.3. Реализация протокола неторопливой свободной состоятельности

При неторопливой свободной состоятельности (LRC) операция синхронизации ***acquire***, открывающая критический интервал, обеспечивает состоятельность всех разделяемых страниц для продолжения выполнения критического интервала. Критический интервал завершается операцией ***release***, после которой всем ВМ становятся доступны переменные, модифицированные в этом критическом интервале. Однако отметки записи для модифицированных разделяемых данных завершаемого критического интервала, необходимые для поддержания состоятельности страниц, могут быть получены другим ВМ только при выполнении им операции ***acquire.***

Когда ВМ выполняет операцию ***acquire*** (s), он посылает ВМ, выполнившему перед ним операцию ***release***, копию своего вектора отметок времени. Этот ВМ в ответ высылает отметки записей, отсутствующие в ВМ, выполняющем операцию ***acquire.*** В пересылаемом списке отметок записи имеются сведения не только о записях, произведенных самим ВМ, но и о записях, о которых ВМ стало известно в результате выполнения им операций синхронизации с другими ВМ.



Так, на рис. 5.5 для трех ВМ в квадратных скобках указаны векторы отметок времени в каждом ВМ. BM_0 и BM_1, выполняют запись в разделяемые страницы памяти в собственных интервалах 2. При операции синхронизации, выполняемой BM_1, BM_1, получает из BM_0, отметку записи в страницу 1. BM_2, при операции синхронизации получает отметки записи в страницы 1 и 2.

Представленный выше механизм обеспечивает поддержку в каждом ВМ списка модифицированных страниц.

После получения отметок записи ВМ запоминает присланные отметки записей и модифицирует вектор отметок времени, присваивая каждой компоненте максимальное значение из собственного вектора и из полученного. Он также делает несостоятельными копии всех страниц, для

которых получены отметки записей, так как эти отметки содержат сведения о модификации страниц после последнего предыдущего согласования копий.

При первом доступе к несостоятельной копии страницы вырабатывается «промах», сигнализирующий о необходимости модификации страницы. Возможны, по крайней мере, два варианта доставки состоятельной страницы. Первый вариант основан на распределенном хранении изменений, внесенных в страницу, второй на поддержании резидентной копии каждой страницы и ее пересылке в затребовавший ВМ. При распределенном хранении изменений при первом доступе к странице по записи создается копия (twin) этой страницы. Далее страница может в течение интервала многократно модифицироваться. При выполнении ***release*** текущее состояние страницы сравнивается с запомненной копией и вычисляется diff совокупность адресов и новых значений, измененных ячеек. ВМ связывает diff с отметкой записи. Когда ВМ нуждается в состоятельной копии страницы, он просматривает имеющийся у него список отметок записей и определяет, какие diff ему необходимы. Далее ВМ запрашивает из других ВМ требуемые ему diff и создает, после их получения, состоятельную копию страницы, применяя diff в требуемом порядке.

При протоколе неторопливой свободной состоятельности diff формируется и передается по явному запросу в отличие от протокола нетерпеливой свободной состоятельности, при котором отметки записи передаются сразу после своего формирования.

Следует отметить, что недостатком протокола LRC служит необходимость «сборки мусора» — удаления невостребованных копий страниц и diff. Необходимо принимать решение о том, до каких пор хранить эти данные.

При поддержании резидентной копии каждой страницы в конце интервала при выполнении ***release*** вычисленные diff для модифицированных страниц высылаются в резидентные для них ВМ. Резидентные ВМ, используя получаемые diff, модернизируют соответствующие страницы. Недостатком этого протокола служит то, что при обращении многих ВМ с запросом копии страницы возникает «узкое горло» при последовательной отсылке страницы каждому ВМ.

5.4.4. Реализация протокола нетерпеливой свободной состоятельности

При этом протоколе записи буферизируются до тех пор, пока не потребуется сделать их доступными при очередном выполнении операции ***release*** путем трансляци онной рассылки измененных данных. Таким образом, основное отличие протокола ERC от LRC состоит в том, что состоятельность страниц поддерживается операцией ***release***, а не ***acquire***. При этом протокол ERC может делать лишние передачи данных для поддержания состоятельности копий страниц, к которым в некоторых ВМ не будет обращений. Соответственно недостатком LRC служит то, что после обращения к несостоятельной странице требуется ожидание, пока будет вычислени передан этому ВМ необходимый diff.

5.4.5. Автоматическое аппаратное распространение записей в удаленные памяти

Существенным недостатком приведенных выше реализаций DSM на базе традиционных систем передачи сообщений служит необходимость программного определения diff. Для преодоления этого недостатка предлагается использовать аппаратные средства удаленной записи в память и удаленного чтения из памяти другого ВМ. Эти средства являются, по своей

сути, ограниченной реализацией рефлексивной памяти. Ниже приводится подход к реализации протокола свободной состоятельности с применением аппаратных средств доступа в удаленную память.

Для эффективной реализации протокола свободной состоятельности важно быстро обнаруживать записи в разделяемые страницы, предотвращать использование несостоятельных данных, получать состоятельные страницы данных при возникновении потребности в них.

Запись в разделяемую страницу обнаруживается механизмом страничной организации памяти при попытке записи в несостоятельную страницу или страницу, доступную только для чтения.

Предотвращение использования несостоятельных данных основывается на отметках логического времени и векторах отметок логического времени.

Получение состоятельных копий страниц вместо применения diff основано на механизме записей в удаленную память и чтения из удаленной памяти.

В каждом ВМ каждой странице сопоставляется вектор отметок времени слива. В этом векторе отмечаются все изменения каждой страницы, которые наблюда ет ВМ.

В каждом ВМ каждой разделяемой странице памяти сопоставляется вектор отметок синхронизации. Этот вектор устанавливает, какие преобразования конкретной страницы необходимы прежде, чем ВМ может получить к ней доступ. Собственно, этот вектор имеет для каждой страницы смысл списка отметок записи протокола LRC. Для ускорения операции acquire каждый ВМ содержит отметку глобального времени синхронизации, имеющую максимальное значение логического времени из всех постраничных отметок этого ВМ.

Если элемент в векторе отметок времени слива страницы меньше, чем соответствующий элемент вектора отметок синхронизации, то соответствующая страница уже не состоятельна. При обратной ситуации, ВМ имеет более новую копию страницы по сравнению с той, которая ему необходима в соответствии с моделью состоятельности памяти.

Протокол **Copyset-2** поддерживает одновременно состоятельными разделяемые страницы не более чем в двух ВМ. Аппаратные средства удаленной записи в память копируют записи в памяти этих ВМ. При возникновении доступа к разделяемой странице со стороны третьего ВМ копия страницы в одном из разделявших ее ранее ВМ объявляется несостоятельной, а другой из этих двух ВМ создает копию страницы для третьего ВМ и устанавливает с ним взаимное удаленное копирование разделяемой страницы.

При выполнении ***release*** ВМ увеличивает значение своего логического времени и использует его значение для удаленного изменения соответствующих элементов векторов отметок времени слива для страниц, которые подверглись модификации в завершаемом интервале.

При инициации выполнения операции ***acquire*** ВМ посылает другому ВМ, последним исполнившему операцию ***release***, свой текущий глобальный вектор отметок времени синхронизации, как это делается в протоколе LRC. Последний, получив этот вектор, устанавливает, какие страницы несостоятельны в запрашивающем ВМ, и высылает отметки записи для всех этих страниц. По получении этих отметок записи ВМ, в продолжение выполнения ***acquire***, модифицирует свой вектор отметок синхронизации, присваивая элементам, соответствующим страницам, для которых поступили отметки записи, максимальное значение логического времени из отметок записи и текущего значения этих элементов.

Если каждый элемент вновь сформированного вектора отметок синхронизации не больше соответствующего элемента вектора отметок времени слива, то ВМ узнает, что у него все страницы состоятельны. В противном случае, если не все стра ницы состоятельны, то несостоятельные страницы помечаются как отсутствующие. Модификация этих страниц еще не завершена, но уже инициирована,

Если ВМ при доступе к странице обнаруживает ее несостоятельность, то он вновь сравнивает элементы вектора отметок синхронизации с элементами вектора отметок времени слива. Если обнаруживается состоятельность соответствующей страницы, то она помечается как состоятельная, и ВМ продолжает выполнение интервала. В противном случае ВМ ожидает, пока страница станет состоятельной, сравнивая элементы вышеназванных векторов.

Протокол **Copyset-N** использует резидентное размещение разделяемых страниц в ВМ. Другие ВМ, имеющие копию страницы, настраивают отображение копии этой страницы в резидентную страницу для автоматического изменения последней при модификации копии.

При выполнении ***release*** ВМ увеличивает значение своего логического времени и использует его значение для удаленного изменения соответствующих элементов векторов отметок времени слива в каждой из резидентных страниц, которые подверглись модификации в завершаемом интервале.

При выполнении операции ***acquire*** ВМ действует так же, как в протоколе **Copyset-2**. Однако он не ждет, пока страницы станут состоятельными, а запрашивает их из резидентных ВМ. При этом запрашивающий ВМ получает также вектор отметок времени слива, что позволяет ему оценить состоятельность поступившей страницы.

Протокол AURC (automatic update release consistency) объединяет оба вышеописанных протокола, работая в случае двух копий как **Copyset-2**, а при большем числе копий как **Copyset-N**.

5.4.6. Направления развития

Актуальность разработки протоколов состоятельности распределенной разделяемой памяти определяется переходом к технологиям InfiniBand и Rapid I/O, которые в своей основе имеют модель распределенной обработки. В рамкак этой модели вычислительные системы предлагается создавать на соединениях «точка-точка», с массовым параллелизмом и коммутацией пакетов. Фактически, для объединения процессоров, памятей внешних устройств предлагается использовать компоненты трех типов: адаптеры с шиной процессора, коммутаторы и адаптеры терминальных устройств.

Технология InfiniBand вводит передачу сообщений на базе операций send/receive. а также операции удаленного доступа в память по записи и чтению RDMA Write, RDMA Read и атомарные операции Compare & Swap and Fetch & Add. Последние используются для синхронизации распределенного протекания процессов.

Поэтому создаваемые программные реализации DSM, направленные на использование сегодняшних кластеров, будут востребованы при построении систем на базе технологий InfiniBand и Rapid I/0.

5.5. Механизм явной реализации когерентности

При явной реализации когерентности используются отдельные наборы команд типа load, store для работы с локальной памятью ВМ и специальные команды (вызовы процедур) типа send, receive для управления адаптерами каналов ввода-вывода. Задача программиста эффективно запрограммировать

передачи данных, совмещая их по возможности с вычислениями и минимизируя объем передаваемых данных.

В связи с тем что внутри микропроцессоров есть несколько уровней кэш-памяти, необходимо учитывать, что вновь прибывшая строка данных делает несостоятельной копию этой строки в кэш-памяти. Поэтому необходимо предусмотреть организацию когерентности прибывшей строки и уже кэшированных строк. Возможны варианты:

- иметь дубликаты тегов строк кэш-памяти в контроле прямого доступа, что позволит делать несостоятельной только действительно необходимую строку;
- не иметь дубликатов тегов и делать по каждому приему строки несостоятельными все строки кэш-памяти.

Использование явной реализации когерентности обусловлено достаточно большими затратами аппаратуры или времени на реализацию неявного механизма когерентности в создаваемой ВС. Это обусловлено тем, что войти внутрь механизма когерентности иерархической памяти сложнее, чем использовать уже предусмотренный для работы внешних устройств механизм когерентности.

Вопросы к главе 5

1. Какой смысл термина «когерентность» применительно к многоуровневой памяти?
2. Как можно реализовать явную когерентность памятей?
3. В чем суть неявной когерентности памятей? Достоинства и недостатки неявной когерентности.

4. Изложите возможную классификацию архитектур вычислительных систем, используя в качестве одного из признаков механизм реализации когерентности многоуровневой памяти.
5. Как реализуется когерентность кэш-памяти и основной памяти?
6. Приведите основные типы организации кэш-памяти.
7. Что такое кэш-строка?
8. Какие алгоритмы используются для определения замещаемых строк?
9. Как можно объяснить существование разных механизмов реализации когерентности многоуровневой памяти? Почему не выбран один, «оптимальный»?
10. В чем состоит проблема реализации когерентности распределенной памяти? Какие существуют модели когерентности памяти?
11. В чем состоит суть аппаратной реализации когерентности многоуровневой памяти?
12. Какие способы реализации когерентности используются обычно в SMP?
13. Что характеризует архитектуру NUMA?
14. Что составляет суть архитектуры COMA?
15. Рассмотрите достоинства и недостатки вышеупомянутых архитектур.
16. Какими свойствами обладает рефлексивная память?
17. В чем состоят особенности программных реализаций разделяемой памяти по сравнению с аппаратными?
18. Объясните суть реализации модели свободной состоятельности разделяемой памяти.
19. Для чего используется вектор отметок времени?
20. Что такое отметка записи?

21. Рассмотрите реализацию протокола неторопливой свободной состоятельности. Какие режимы работы с памятью эффективно реализуются?
22. Чем отличается реализация протокола нетерпеливой свободной состоятельности от протокола неторопливой свободной состоятельности?
23. Какие преимущества имеет реализация протокола нетерпеливой свободной состоятельности, использующая автоматическое аппаратное распространение записей в удаленные памяти?