Московский Государственный Университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

**С.В.Герасимов, И.В.Машечкин, М.И.Петровский,
И.С.Попов, А.Н.Терехин, А.В.Чернов**

Организация кэширования

(учебно-методическое пособие)

Москва
2011

УДК 004.254
ББК 32.973-018.2

С40

Пособие посвящено базовым аспектам реализации процесса кэширования в современных вычислительных архитектурах. В пособии рассматриваются вопросы работы ассоциативного кэш и кэш прямого доступа, алгоритмов вытеснения, стратегии записи, организации многоуровнего кэш. Отдельное внимание уделено оценкам производительности кэш, методам повышения производительности, а также организации кэширования в многопроцессорных архитектурах типа UMA и NUMA. Пособие издано в поддержку курса "Операционные системы", читаемого студентам второго курса на факультете ВМК МГУ имени М.В. Ломоносова. Материал рассчитан на широкий круг читателей: студентов, аспирантов, преподавателей, а также всех интересующихся.

УДК 004.254
ББК 32.973-018.2

Рецензенты:

доцент И.А.Волкова
доцент Е.А.Кузьменкова

С.В.Герасимов, И.В.Машечкин, М.И.Петровский, И.С.Попов, А.Н.Терехин, А.В.Чернов

С40 **Организация кэширования: учебно-методическое пособие**.

Издательский отдел факультета ВМК МГУ 2011, - 26 с.

Печатается по решению Редакционно-издательского Совета факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова

ISBN 978-5-89407-468-9

Основы кэширования

Понятие кэширования

Одной из проблем производительности компьютеров является несоответствие скорости работы центрального процессора и скорости доступа к информации, размещенной в оперативной памяти. Процессору приходится простаивать при обращении к оперативной памяти в ожидании получения или записи данных. Причем тенденция увеличения скорости работы центрального процессора и увеличения скорости работы ОЗУ говорит о том, что данная проблема будет в будущем увеличиваться, поскольку с момента появления вычислительной техники производительность центральных процессоров росла экспоненциально, а скорость доступа ОЗУ линейно.

Для сглаживания данной проблемы используется **Кэш** (от фр. cacher — «прятать») — специальный промежуточный буфер высокоскоростной памяти небольшого размера. Такой буфер содержит ту часть информации из ОЗУ, которая потенциально может быть запрошена с наибольшей вероятностью. Это достигается за счет использования **принципа локальности программ,** суть которого состоит в предположении о том, что большинство программ в каждый момент времени использует те же участки кода и данных, которые использовались недавно, и, таким образом, не обращаются случайно и равномерно ко всему коду и всему объему данных.

Кэш состоит из набора блоков памяти, каждый из которых ассоциирован с блоком данных в основной памяти. Каждый блок имеет идентификатор - **тэг**, определяющий соответствие между блоками данных в кэш и их копиями в основной памяти. Таким образом, кэш является частным случаем **ассоциативной памяти**, т.е. памяти адресуемой по содержимому. При использовании кэш обмен данными при выполнении программы (чтение команд, чтение значений операндов, запись результатов) происходит не с ячейками оперативной памяти, а с кэш. Когда центральный процессор обращается к данным сначала проверяется кэш. Если в нем необходимые данные присутствуют, т.е. найден блок, совпадающий с блоком затребованного элемента данных в ОЗУ, то используется информация в кэш. Такой случай называется **попаданием кэш**. Если в кэш не найдена необходимая информация, то содержащий ее блок данных читается из основной памяти в кэш, и становится доступным для последующих обращений. Такой случай называется **промахом кэш**. Процент обращений к кэш, когда в нём найден результат, называется **уровнем попаданий** или **коэффициентом попаданий** в кэш. Поскольку кэш имеет размер заметно меньший по сравнению с ОЗУ, то при промахе нужно отбросить некоторый блок для освобождения пространства.

Для выбора отбрасываемого блока используются алгоритмы **вытеснения**. При записи данных в кэш выполняется их обновление в основной памяти. Порядок во времени между изменением данных в кэш и обновлением данных в ОЗУ определяется **стратегией записи.**

Ассоциативность кэш, организация адресации

Существуют три основных подхода к организации отображения блоков оперативной памяти в кэш:

- *Полностью ассоциативный кэш*. Блок ОП может быть размещен в любой позиции в кэш.
- *Кэш прямого отображения* (direct mapping). В этом случае каждый блок ОП может быть размещен в единственной позиции в кэш, заданной формулой:

$$A_{кэш} = A_{ОП} \text{ MOD } N_{кэш}$$

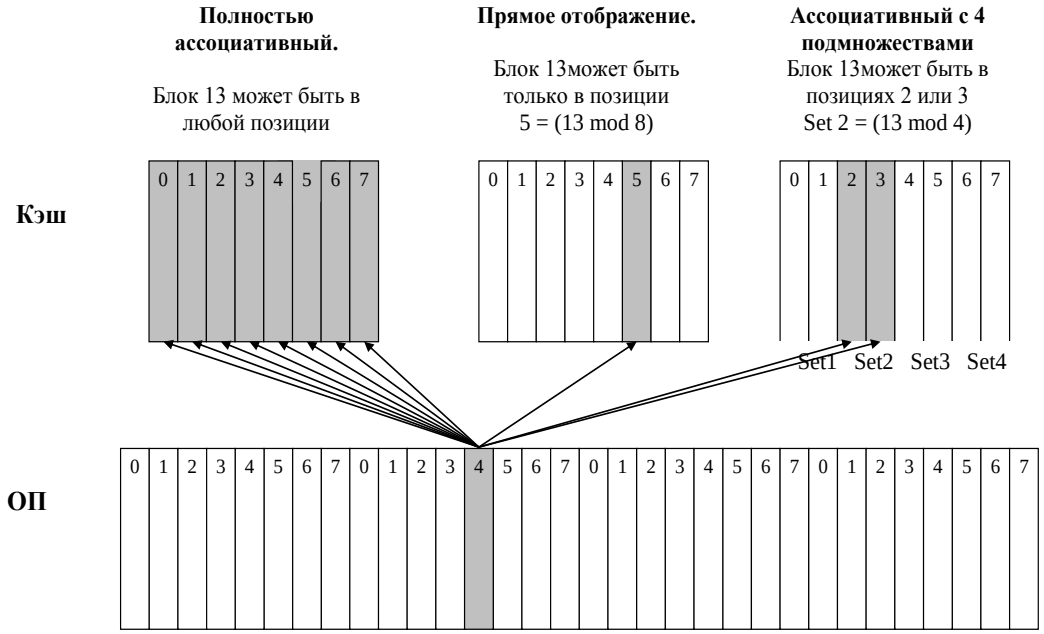
$A_{кэш}$ – адрес блока в кэш,
$A_{ОП}$ - адрес блока в ОП,
$N_{кэш}$ - число блоков в кэш.

- *Частично ассоциативный кэш*. В этом случае множество блоков кэш разделяется на несколько непересекающихся подмножеств, и блок ОП может быть отображен в любой блок кэш из строго заданного подмножества:

$$A_{п/м\ кэш} = A_{ОП} \text{ MOD } N_{п/м\ кэш}$$

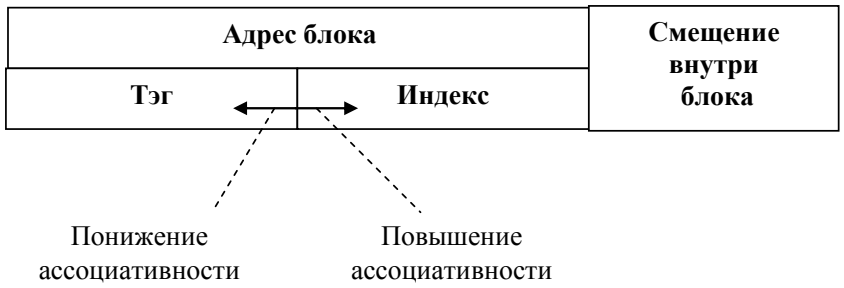
$A_{п/м\ кэш}$ – адрес подмножества блоков в кэш,
$A_{ОП}$ - адрес блока в ОП,
$N_{п/м\ кэш}$ – число подмножеств блоков в кэш.

Данные подходы представлены на рисунке.



Блок 13

Если число подмножеств частично ассоциативного кэш обозначить *m*, а число блоков в нем *n*, то такой кэш называют *m-ассоциативный кэш*. Таким образом, прямое отображение и полностью ассоциативный кэш являются частными случаями частично ассоциативного кэш (1- и n- ассоциативные кэш соответственно).



Адрес в кэш содержит две позиции: **смещение** в блоке и **тэг.** В случае частично ассоциативного кэш добавляется еще позиция **индекс,** который нужен для выбора подмножества в частично ассоциативном кэш. Тэг – уникальный для каждого блока спецификатор. В теге находится служебная информация, характеризующая данный блок кэш. В нем может содержаться информация о том, какому блоку ОП соответствует содержимое данного блока кэш, занят или свободен блок, производились изменения в данном блоке или нет, и другая информация. Когда процессору нужно обратиться за командой или за данными в ОП, сначала происходит обращение к кэш, и поиск по всем тегам происходит параллельно в рамках подмножества тегов, заданного индексом. Следует отметить, что увеличение

степени ассоциативности частично ассоциативного кэш увеличивает число блоков в подмножестве, таким образом, уменьшая размерность индекса в адресе и увеличивая размер тэга. Полностью ассоциативный кэш не имеет поля индекс, а вся информация содержится в теге. Но в этом случае и поиск осуществляется дольше, поскольку происходит по всем тегам, а не по определенному их подмножеству. С другой стороны, в кэш прямого отображения индексом однозначно определяется номер блока в кэш, а тэг содержит только информацию о состоянии этого блока.

Анализ тэгов блоков кэш производится аппаратно. После вычисления исполнительного адреса операнда или команды устройство управления может определить, находится ли соответствующая информация в одном из блоков кэш-памяти и является она актуальной или нет.

Алгоритмы вытеснения

При возникновении промаха происходит **вытеснение** – обновление содержимого кэш. Для этого в кэш выбирается блок-претендент на вытеснение, т.е. блок, содержимое которого будет заменено на содержимое нового блока из ОП. Стратегия этого выбора зависит от конкретной организации процессора и от используемого подхода к отображению. В частности, в кэш прямого отображения определение кандидата на вытеснение происходит быстро и однозначно, поскольку адрес вытесняемого блока однозначно связан с адресом загружаемого блока.

Для частично или полностью ассоциативных кэш вытеснение блоков может осуществляться одним из следующих способов:

- *случайный выбор* - номер вытесняемого блока определяется встроенным генератором случайных чисел в соответствии с равномерным распределением;
- *LRU* (Least-Recently Used) - вытеснение *неиспользуемого дольше всех* блока, в этом случае сохраняется информация о «возрасте» блока кэш, т.е. метки последнего обращения, причем каждое обращение влечет пересчет «возраста» для всех блоков;
- *LFU* (Least Frequently Used) – вытеснение *наиболее редко используемого* блока, аналогично LRU, но при этом считается и хранится число обращений к блоку, что, в отличие от LRU, не требует обновления информации во всех блоках при каждом обращении;
- *MRU* (Most-Recently Used) – достаточно экзотический подход, при котором происходит вытеснение *последнего использованного блока*, т.е. стратегия обратная LRU, эффек-

тивна при обработке программой последовательно больших объемов памяти;

- *Комбинированные методы* – могут сочетать случайный поиск с LRU и LFU.

Основное достоинство метода случайного вытеснения заключается в простоте его аппаратной реализации. С увеличением степени ассоциативности алгоритмам LRU, LFU и MRU требуется все больше времени для поиска блока-кандидата на вытеснение. Поэтому для частично или полностью ассоциативных кэш большого объема, как правило, используются методы случайного вытеснения или приближенные алгоритмы типа LRU, в которых выбирается не самый редко используемый блок, а один из нескольких редко используемых, возможно не самый редкий.

Стратегии записи

Отдельно следует обратить внимание на организацию работы с кэш при записи данных. В целом, количество операций чтения данных преобладает, а в случае использования кэш команд чтение – единственная операция. Поэтому основная оптимизация производительности кэш ориентирована на повышения эффективности чтения. В частности, для операций чтения данные из блока кэш могут считываться параллельно с анализом тега этого блока. Если анализ показывает, что произошло попадание, то экономия времени возникает за счет параллельного считывания. Если же произошел промах, то считанные данные просто игнорируются. При этом выигрыша по скорости нет, но нет и проигрыша по сравнению с подходом, при котором анализ тэга и считывания блока происходят строго параллельно. Но такой подход не годится для операций записи, поскольку изменение данных можно производить только в случае попадания в правильный блок, т.е. строго после завершения анализа тэга. Кроме того, обычно операция записи изменяет не весь блок, а одно машинное слово или меньше. Поэтому операции записи требуют большего времени.

Существуют две стратегии организации записи при использовании кэш:

- **сквозная запись** – информация пишется сразу и в кэш, и в ОП;
- **отложенная запись** – информация пишется только в кэш, данный блок помечается специальным флагом, а при выполнении вытеснения обновленный блок передается в ОП целиком.

Оба подхода имеют свои достоинства и недостатки. В частности, при отложенной записи число обращений к оперативной памяти уменьшается за счет аккумуляции всех изменений блока до момента его вытеснения из кэш. С другой стороны, при сквозной записи операции чтения не приводят к необходимости записи блока в оперативную память, а данные в оперативной памяти всегда консистентны, что особенно важно для архитектур с параллельными вычислениями – многоядерных и многопроцессорных.

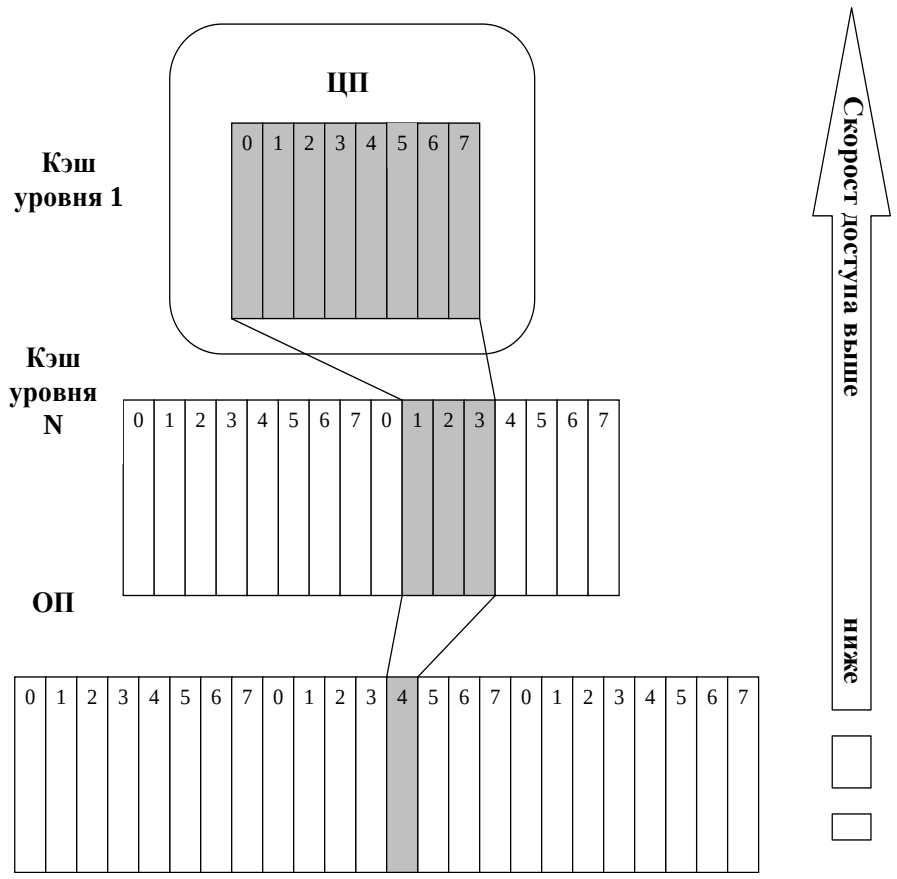
Еще одной особенностью организации кэш является алгоритм работы в случае промахапри операции записи, т.е. отсутствия в кэш блока памяти в который осуществляется запись. В этом случае возможны два подхода:

- *загрузка при записи* (fetch on write) - загрузка из оперативной памяти в кэш блока, в который производилась запись;
- *прямая запись* (write around) – в этом случае загрузка блока в кэш не производится.

Любой из данных подходов может использоваться совместно с любой из стратегий записи, но обычно стратегия «сквозная запись» используется с алгоритмом «прямая запись», а «отложенная запись» совместно с «загрузкой при записи».

Многоуровневый кэш

Дальнейшим развитием подхода кэширования является организация **многоуровневой** или **иерархической** структуры кэш памяти. В этом случае добавляется несколько промежуточных уровней кэш памяти между процессором и основной оперативной памятью. Поскольку аппаратная реализация быстрой памяти на единицу хранимой информации дороже, чем аппаратная реализация более медленной, то размер памяти каждого уровня увеличивается по направлению от процессора к оперативной памяти, а скорость доступа, соответственно, убывает. При этом, как правило, уровни вложены, т.е. данные, содержащиеся в более быстром и менее объемном уровне, содержатся и в следующем, более медленном и более объемном.



Как и в случае одноуровневого кэш, эффективность многоуровневой организации обеспечивается принципом локальности программ. При этом создается эффект того, что вся память доступна со скоростью самого быстрого, нижнего уровня иерархии, и имеет объем самого большого уровня иерархии. Важным моментом при организации многоуровневых кэш является стратегия размещения одних и тех же блоков в разных уровнях. Такое размещение может быть:

- *инклюзивным* – каждый блок нижнего уровня содержится в блоке верхнего, и при вытеснении происходит замещение блоков по всей цепочке уровней;
- *эксклюзивным* – блок, находящийся в одном из уровней, гарантированно отсутствует в других, при этом операция вытеснения менее трудоемка;
- *смешанным* – жестких правил размещения блоков нет, вытеснение происходит в соответствии с работой алгоритмов вытеснения каждого из уровней.

Производительность кэш и методы ее повышения

Использование кэш-памяти позволяет повысить эффективность работы с оперативной памятью. Во-первых, сокращается количество обращений к ОЗУ, как по выборке команд, так и по выборке операндов. Во-вторых, существенно увеличивается скорость доступа к памяти в случае использования ОП с «расслоением», так как обмены блоков с памятью проходят практически параллельно (когда мы работаем с группой подряд идущих слов). В-третьих, поскольку размещение и команд, и данных в одном кэш может приводить к тому, что блоки кода и данные начинают вытеснять друг друга, увеличивая при этом обращения к оперативной памяти, современные компьютеры имеют два независимых кэш: **кэш данных** и **кэш команд,** каждый из которых работает со своим потоком информации, т.е. первых хранит только блока памяти с данными, второй только с кодом.

Естественно, при использовании кэш возникают и проблемы, связанные с усложнением логики работы процессора: при выборке очередных команд, получении операндов команд и записи результатов выполнения команд в ОЗУ добавляются схемы организации использования кэш-памяти. Производительность кэш обычно оценивается по следующей формуле, где элементы формулы могут оцениваться либо в абсолютных временных единицах, например, наносекундах, либо в циклах ЦП:

$$T_{ОП} = T_{кэш} + P_{промах} \times C_{промах}$$

$T_{ОП}$ – среднее время доступа к ОП,
$T_{кэш}$ - время доступа к кэш,
$P_{промах}$ – вероятность промаха в кэш,
$C_{промах}$ – стоимость промаха в кэш.

В формуле есть три составляющие $T_{кэш}$, $P_{промах}$, $C_{промах}$, за счет уменьшения которых можно повышать эффективность работы кэш. Кратко рассмотрим некоторые популярные методы повышения эффективности кэш. Следует отметить, что стратегии увеличения производительности кэш, как правило, ориентированы на оптимизацию по одному из трех параметров формулы, при этом бывает, что другие параметры могут ухудшаться. Кроме того, повышение эффективности работы кэш может достигаться как за счет аппаратных решений, так и программных. Пользователь компьютера или программист может (и должен) использовать программные средства, на аппаратные он влиять не может. Далее рассмотрим некоторые показа-

тельные и простые программные и аппаратные методы повышения эффективности работы с кэш.

Программные методы повышения эффективности работы кэш

Большинство программных средств повышения эффективности работы с кэш сводятся к тому, чтобы за счет правильной организации использования программой алгоритмов и структур данных в ней выполнялся *принцип локальности*. Примером таких средств способов организации работы программы могут служить *объединение массивов* и *объединение циклов*.

Многие программы требуют одновременного обхода двух и более массивов по одному общему индексу. Это создает угрозу того, что обращения к разным массивам будет приводить к конфликтам и постоянному вытеснению и возврату блоков в кэш. Например:

```
//до оптимизации

int a[100000];
int b[100000];
int c[100000];


for (int i = 0; i < 100000; i++)
{
     c[i] = a[i]+b[i];
}
```

Поэтому целесообразно объединять массивы в общий массив структур.

```
//после оптимизации

struct i2
{
     int a;
     int b;
}

i2 ab[100000];
int c[100000];


for (int i = 0; i < 100000; i++)
{
     c[i] = ab[i].a+ab[i].b;
}
```

Часто программа имеет отдельные участки кода для доступа к одним и тем же массивам по одним и тем же индексам, но с различными вычислениями внутри цикла. Например:

```
//до оптимизации

for (int i = 0; i < 100000; i++)
{
     for (int j = 0; j < 100000; j++)
     {
          a[i][j] = b[i][j]*c[i][j];
     }
}

for (int k = 0; k < 100000; k++)
{
     for (int l = 0; l < 100000; l++)
     {
          d[i][j] = a[i][j]+c[i][j];
     }
}
```

В этом случае, за счет объединения вычислений в общий цикл, данные могут повторно использоваться (считываться) из кэш до их записи в случае кэш с отложенной записью.

```
//после оптимизации

for (int i = 0; i < 100000; i++)
{
     for (int j = 0; j < 100000; j++)
     {
          a[i][j] = b[i][j]*c[i][j];
          d[i][j] = a[i][j]+c[i][j];
     }
}
```

Существуют различные программные возможности повышения эффективности использования кэш, в том числе, встроенные в компиляторы. Но для правильного использования средств оптимизации программ важно знать, какая именно организация кэш имеется на целевой архитектуре. В частности, последний пример с объединением циклов не даст положительного эффекта в случае сквозной записи, поскольку потребует выгрузки блока из кэш в оперативную память после каждой операции записи.

Аппаратные методы повышения эффективности работы кэш

Самым простым методом повышения производительности кэш за счет *понижения вероятности промаха* является *увеличение размера блока памяти*. Действительно, чем больше блок, тем выше вероятность того, что программа последовательно будет обращаться к одному и тому же блоку памяти, если выполняется принцип локальности. Кроме того, чем больше размер блока, тем меньше нужно блоков для того же суммарного объема памяти, а, значит, быстрее будут работать алгоритмы поиска, что также несколько *уменьшает время доступа к кэш*. Но, с другой стороны, с увеличением размера блока возрастают накладные расходы на его вытеснение, что может привести к сильному увеличению *стоимость промаха*.

Наиболее популярным методом понижения вероятности промаха является *повышение ассоциативности кэш*. Чем выше ассоциативность, тем больше возможностей у алгоритма кэширования при вытеснении оставить нужный блок, который будет использоваться в ближайшем времени. Но с другой стороны, как уже обсуждалось выше, при высоком уровне ассоциативности, неслучайные алгоритмы поиска становятся вычислительно сложными, что значительно увеличивает *время доступа к кэш*.

Одним из подходов к понижению стоимости промаха за счет аппаратных решений является организация так называемого *не блокирующего при промахе кэш*. Он может быть реализован в случае использования параллельных конвейерных процессоров, которые позволяют командам завершаться не в том порядке, в котором они запускались, если между ними нет зависимостей по результату. Тогда процессор может не останавливаться в случае промаха в кэш, а продолжать вычисления потока инструкций, если они не требуют обращений к тому же блоку памяти. При этом значительно усложняется логика работы кэш, поскольку ему необходимо параллельно обрабатывать процедуру вытеснения блока после промаха и процедуру обращения к другим блокам. Большинство процессоров, работающих по такой схеме, могут параллельно обрабатывать только один промах с множеством последующих попаданий. Если происходит второй промах, то вычисления останавливаются и обрабатываются все процедуры вытеснения. Но есть архитектуры, работающие с несколькими промахами одновременно. Следует отметить, что такое решение естественно увеличивает *время доступа* к кэш за счет использования более сложных алгоритмов поиска и вытеснения.

Наиболее популярным методом понижения стоимости промаха является *увеличение числа уровней кэш*. В этом случае стоимость промаха понижается за счет того, что обращения к более высокому уровню кэш, хотя и дольше чем обращение к кэш первого уровня, но

значительно быстрее, чем к ОП. Большинство современных компьютеров имеет два и более уровней кэш. Как правило, первые два уровня кэш находятся внутри ЦП, при этом их объем и скорость доступа к ним могут отличаться на порядок. Для *многоядерных* процессоров каждое из ядер, по сути, представляет собой отдельный процессор со своими кэш первого и второго уровней. Кэш третьего и четвертого уровней используются уже на мощных многопроцессорных серверах, мейнфреймах, специализированных вычислительных системах и находятся на отдельной от ЦП микросхеме, обеспечивающей поддержку необходимых алгоритмов доступа и синхронизации. Поскольку уровни кэш могут быть достаточно разнородными, т.е. иметь разные алгоритмы кэширования и даже разный размер блоков, задача разработки эффективных архитектур с многоуровневым кэш и сами эти архитектуры становятся весьма сложными, причем почти всегда значительное уменьшение стоимости промаха за счет многоуровневости негативно сказывается на времени доступа и вероятности промаха.

Самым очевидным решением для сокращения времени доступа является использование кэш с простой организацией и небольшим объемом. Поэтому кэш первого уровня делают всегда небольшим. Но упрощение и уменьшение размера всего кэш естественно увеличивает вероятность и стоимость промаха.

Кэширование в многопроцессорных архитектурах

Все составляющие процесса кэширования - размещение, адресация, вытеснение и запись - могут существенно влиять на возможность работы в случае параллельных многопроцессорных архитектур. Рассмотрим основные подходы к организации кэширования для систем с общей и распределенной памятью. В рамках этих подходов ОП является *разделяемым ресурсом* физически (для общей памяти) или логически (для распределенной). В любом случае возникает проблема *синхронизации* содержимого локальных кэш отдельных процессоров и разделяемой ими ОП. Для описания целостного непротиворечивого состояния локальных кэш используется термин **когерентность кэш**. Когерентность определяет порядок чтений и записей по одному и тому же физическому адресу ОП. Кэш называется когерентным, если выполняются следующие условия:

- *Когерентность выполнения.* Если процессор записывает значение по некоторому физическому адресу в ОП, то при следующем считывании он должен получить ранее записанное значение, если между записью и чтением другой процессор не производил запись по этому адресу.
- *Когерентной видимости.* Операция чтения процессором, следующая после того, как другой процессор осуществил запись по некоторому физическому адресу ОП, должна вернуть записанное значение, если другие процессоры не изменяли эту ячейку памяти между двумя операциями.
- *Когерентность записи.* Записи в одну и ту же ячейку ОП должны быть последовательными, т.е. если два процессора записывают по некоторому адресу в ОП разные значения в определенном порядке, этот порядок должен сохраняться при считывании любым из процессоров.

Для кэш существуют следующие основные подходы поддержания когерентности:

- *Справочник* (directory). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по вычислительным узлам многопроцессорной системы).
- *Отслеживание* (snooping). Каждый кэш, который содержит копию данных некоторого блока физической памяти,

имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэш расположены на общей (разделяемой) шине, и контроллеры всех кэш наблюдают за шиной для определения того, не содержат ли они копию блока, с которым производится операция в настоящий момент.

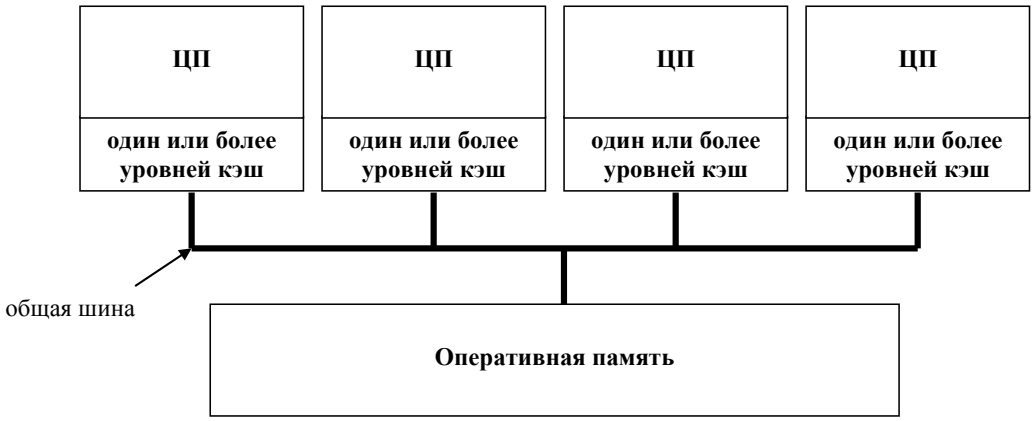
Сами процедуры (последовательности действий) поддержания когерентности называются *протоколами когерентности*. Рассмотрим более детально организацию когерентных кэш для некоторых параллельных архитектур на основе ***общей памяти***. Традиционно по способу организации работы с памятью параллельные архитектуры разделяют на системы:

- *с общей памятью*;
- на основе *передачи сообщений*.

В первом случае все адресное пространство для всех вычислительных узлов параллельной системы представляется единым виртуальным адресным пространством. Это выполнено даже в случае *распределенной общей памяти,* когда каждый вычислительный узел имеет свой физически отдельный модуль памяти и предоставляет его полностью или частично в общее пользование. Во втором случае – *передачи сообщений* - каждый вычислительный модуль функционирует как отдельный компьютер, а весь обмен данными происходит на уровне прикладных программ с помощью аппарата передачи сообщений. Вопрос когерентности кэш для параллельных систем передачи сообщений не актуален, поскольку каждый процесс общей параллельной программы работает в своем изолированном, физически локальном адресном пространстве. В то же время, традиционные алгоритмы решения современных математических задач предполагают массовую обработку больших объемов данных с использованием математических вычислений, организованных в виде последовательностей вложенных циклов. Эффективный перенос таких алгоритмов на системы передачи сообщений достаточно затруднителен, поэтому параллельные системы с общей памятью являются важнейшим компонентом современных вычислительных систем. Для них вопрос синхронизации кэш становится особенно актуальным, поскольку в таких системах оперативная память является физически глобальным разделяемым ресурсом, а память кэш, наоборот, физически локальна и используется только одним процессором.

Кэширование в UMA системах

Рассмотрение ***систем с общей оперативной памятью*** начнем с систем с однородным доступом к памяти (**UMA** - uniform memory access) . В данной модели произвольный процессорный элемент имеет доступ к произвольной точке оперативной памяти (доступ с одинаковым временем). Поэтому характеристики доступа любого процессорного элемента в любую точку ОЗУ не зависят от конкретного элемента и адреса, все процессоры равноценны относительно доступа к памяти. Наиболее популярным представителем UMA-систем является симметричная мультипроцессорная система (**SMP** - symmetric multiprocessor).



В этой модели к общей системной шине, или магистрали, подсоединяются несколько процессоров и блок общей оперативной памяти. У данного решения можно отметить следующие *недостатки*. Во-первых, это централизованная система, и общая шина в ней является «узким горлом», поэтому данная модель накладывает существенные ограничения на количество подключаемых процессорных элементов (обычно 2, 4, 8, вплоть до 32). Во-вторых, возникают проблемы синхронизации кэш каждого процессора.

В силу особенностей реализаций основным методом организации когерентных кэш для UMA систем является *отслеживание* (snooping), поскольку в любом случае в системе существует общая высокоскоростная шина и любые операции работы с оперативной памятью проходят через нее. Проблема когерентности кэш возникает только при операции записи. Существуют два базовых протокола поддержания когерентности:

- протокол **инвалидации кэш при записи;**
- протокол **широковещательной записи.**

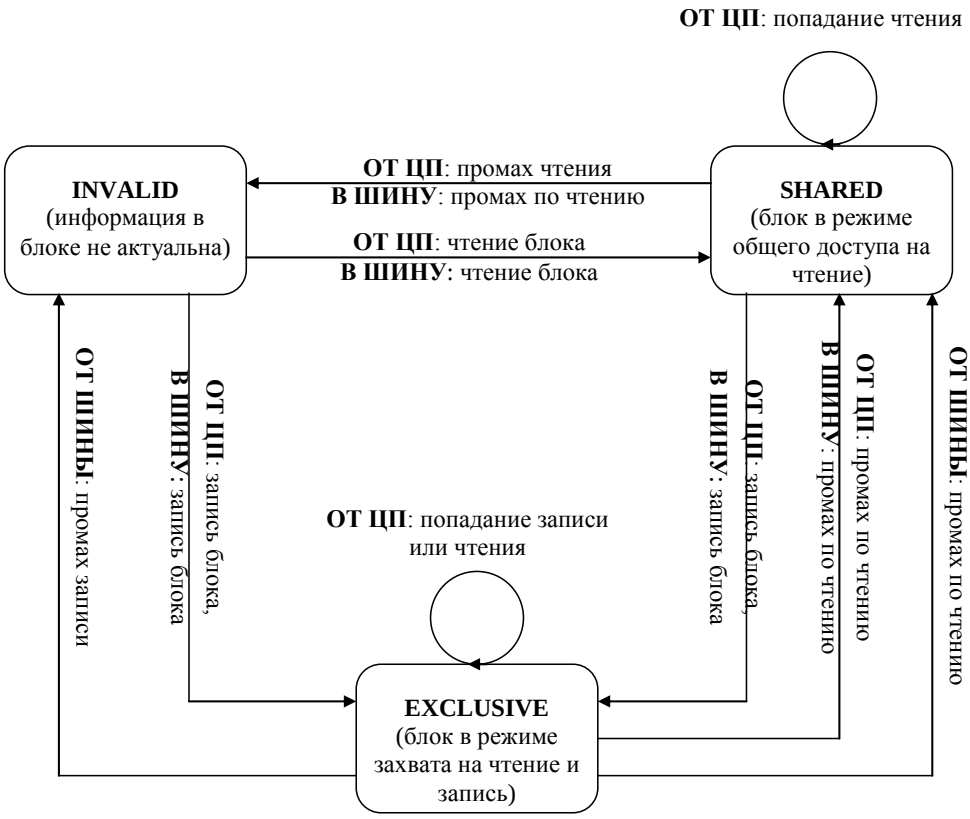
Инвалидация при записи является наиболее часто используемым протоколом как для подхода *отслеживания* (snooping) так и

для подхода на основе *справочника*. В нем для проведения операций записи процесс захватывает *эксклюзивно* соответствующий блок памяти. В результате этого все кэш блоки, содержащие данный блок памяти, становятся *невалидными*. Их тэги в локальных кэш проставляют специальный флаг, а при попытке обратиться с операцией чтения или записи к данному блоку другим процессом произойдет промах кэш, и данный блок будет заново подгружен из оперативной памяти после освобождения эксклюзивного захвата. Нетрудно проверить, что все правила когерентности при таком протоколе выполняются.

Альтернативой рассмотренного протокола является протокол, при котором происходит непосредственная запись обновленного блока данных в кэш всех процессоров, где он используется. Такое действие называется *широковещательной записью.* Этот протокол используется достаточно редко, поскольку он сложен в реализации и обладает существенным недостатком. В случае последовательных операций записи одним процессором в один блок памяти без одновременного чтения другими процессорами данных протокол широковещательной записи потребует обновления всех кэш после каждой операции, в то время как протокол инвалидации просто проставит соответствующий флаг. С другой стороны, время реального обновления данных в кэш всех процессоров, использующих измененный блок памяти, естественно меньше при использовании широковещательной записи, поскольку операция обновления кэш работает быстрее, чем последовательность операций: инвалидация, промах, замещение.

При использовании протоколов когерентности кэш на основе *отслеживания* в UMA системах все процессоры постоянно просматривают появляющиеся операции работы с памятью через общую шину и проверяют, происходит ли работа с теми же блоками памяти, что находятся у них в локальных кэш, проставляя соответствующий признак невалидности или обновляя данные в кэш. В случае использования кэш с сквозной записью функционирование таких систем очевидно. В случае же кэш с отложенной записью ситуация сложнее, поскольку результаты самого последнего изменения могут находиться не в оперативной памяти, а в локальном кэш одного из процессоров, к которому нет доступа у других процессоров. В этом случае обращение к памяти будет прервано до тех пор, пока не произойдет выгрузка соответствующего блока кэш, владеющим им процессором. Вся информация о валидности, эксклюзивном захвате с указанием процессора-*владельца* и разделяемом доступе на чтение без захвата хранится в тэг кэш каждого процессора и обновляется в зависимости от действий ЦПУ или сигналов из шины. Модельное

представление переходов состояния отдельного блока кэш в отдельном локальном процессоре показано на диаграмме состояний.



Для каждого перехода:

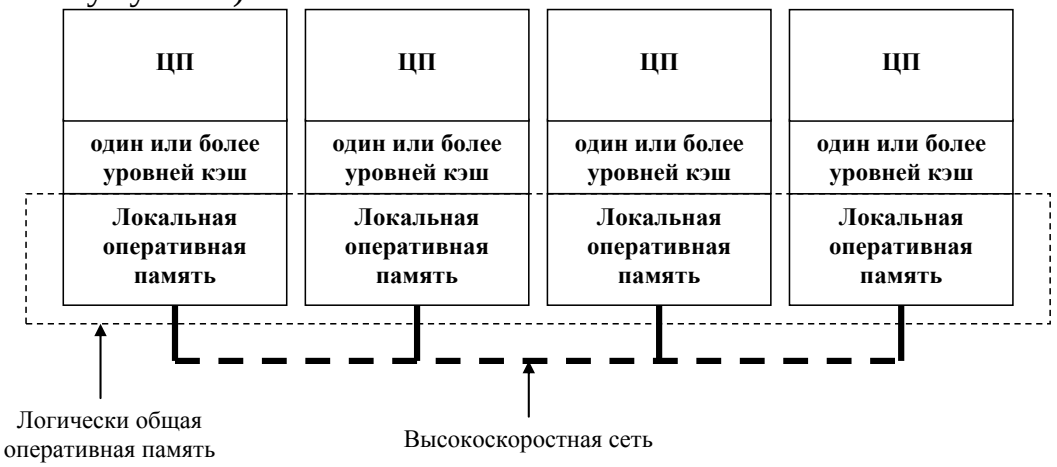
- в разделе «ОТ» указано, кто (ЦП или ШИНА) инициировал переход (от кого пришел сигнал) и за счет какого именно действия (запись, чтение, промах записи, промах чтения);
- в разделе «В» указано, какое сообщение посылается в шину при переходе в целевое состояние.

Примерно такой конечный автомат реализован в кэш контроллере для каждого из кэш блоков. Следует заметить, что смена состояний, в том числе выход из эксклюзивного состояния, происходит автоматически при появлении соответствующих событий от шины или ЦП и не требует «разрешения» у ЦП – «владельца блока». Если два процессора одновременно пишут в один и тот же блок памяти, то они будут конкурировать за «эксклюзивное» состояние этого блока, перехватывая его друг у друга, поскольку «удержать» его невозможно. Атомарность операций и переходов обеспечивается за счет общей шины.

Кэширование в NUMA системах

Иной подход к реализации систем с общей оперативной памятью предлагает архитектура систем с неоднородным доступом к памяти (**NUMA** - non-uniform memory access). Такие архитектуры

также называют системами с распределенной памятью (Distributed memory systems).



Масштабируемость и степень параллелизма в NUMA-системах выше, чем в UMA. Для NUMA-систем характерны следующие свойства:

- процессорные элементы физически являются отдельными вычислительными модулями со своими ЦП, кэш, ОП, контроллером ввода-вывода и т.д., но работают на общем адресном пространстве;
- характеристики доступа процессора к области оперативной памяти зависят от того, к каким областям идет обращение (к локальной или нелокальной памяти).

Контроллер памяти позволяет осуществлять взаимодействие между вычислительными узлами. Доступ процессора к своей ОП осуществляется через свой контроллер, к чужой – через два контроллера, свой и чужой. При обращении не к своей памяти контроллер выбрасывает запрос на общую шину, целевой контроллер его принимает и возвращает результат.

В NUMA-системах остается проблема синхронизации кэш. Существует несколько способов ее решения:

- не использовать кэширование (процессоры без кэш);
- использовать модель ccNUMA (Cache coherent NUMA).

Для синхронизации кэш в ccNUMA системах, как правило, используется подход на основе *словарей*, согласно которому с каждым блоком физически распределенной оперативной памяти связана отдельная запись в словаре. При этом каждый блок и, соответственно, запись в словаре уникальны в рамках системы, а сам словарь может храниться и обрабатываться распределено. Как и в подходе с *отслеживанием*, подход на основе словарей должен обрабатывать две основные ситуации, требующие синхронизации: запись в общую область памяти и промах по чтению.

Упрощенная модель протокола поддержания когерентности определяет для каждого блока памяти в словаре три состояния:

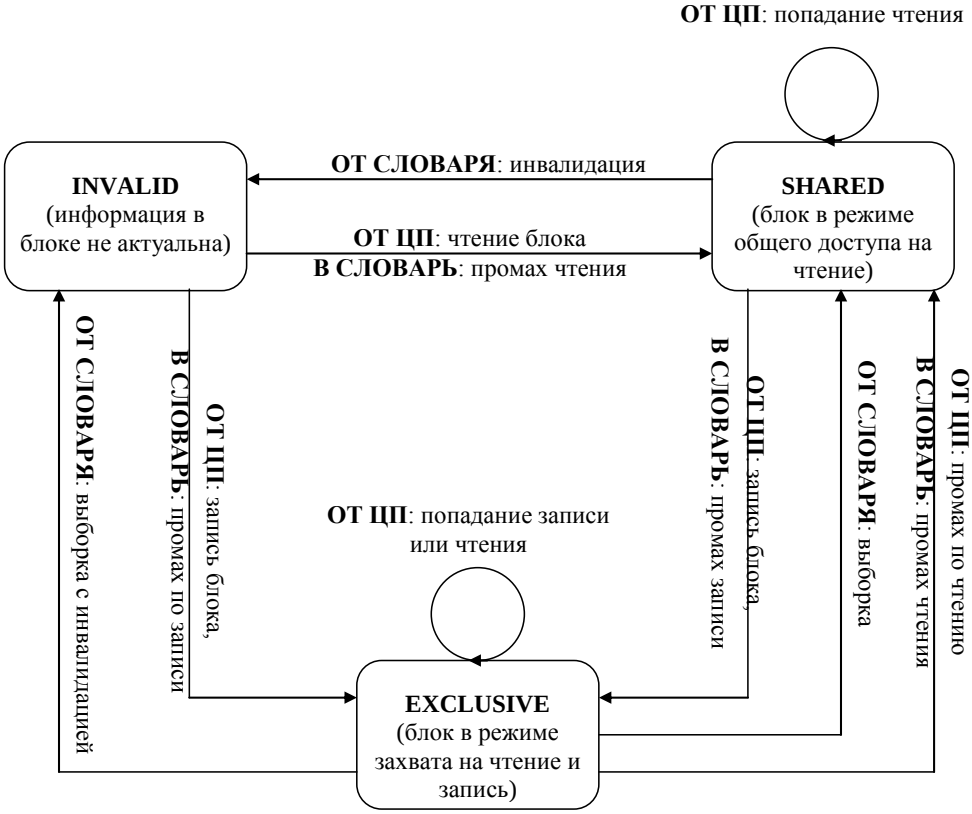
- *Общий доступ* (на чтение) - один или более процессоров загрузили блок к себе в кэш, и данные в этом блоке и во всех кэш консистентны;
- *Не закеширован* – ни один процессор не использует данный блок;
- *Эксклюзивный захват* – один из процессоров захватил блок с целью записи данных, остальные копии в кэш других процессоров неактуальны.

Суть протокола состоит в организации корректных переходов в рамках перечисленных состояний и передачи актуальной информации (копии) блоков память в локальные кэш. Для этого используется механизм передачи сообщений. Рассмотрим, какие типы сообщений могут передаваться. В этом процессе могут участвовать до трех физически отдельных вычислительных узлов: узел Q - физический владелец блока ОП A, узел P, на котором потребовалось изменить данные D в блоке A, и узел R – физический владелец записи в словаре о блоке A.

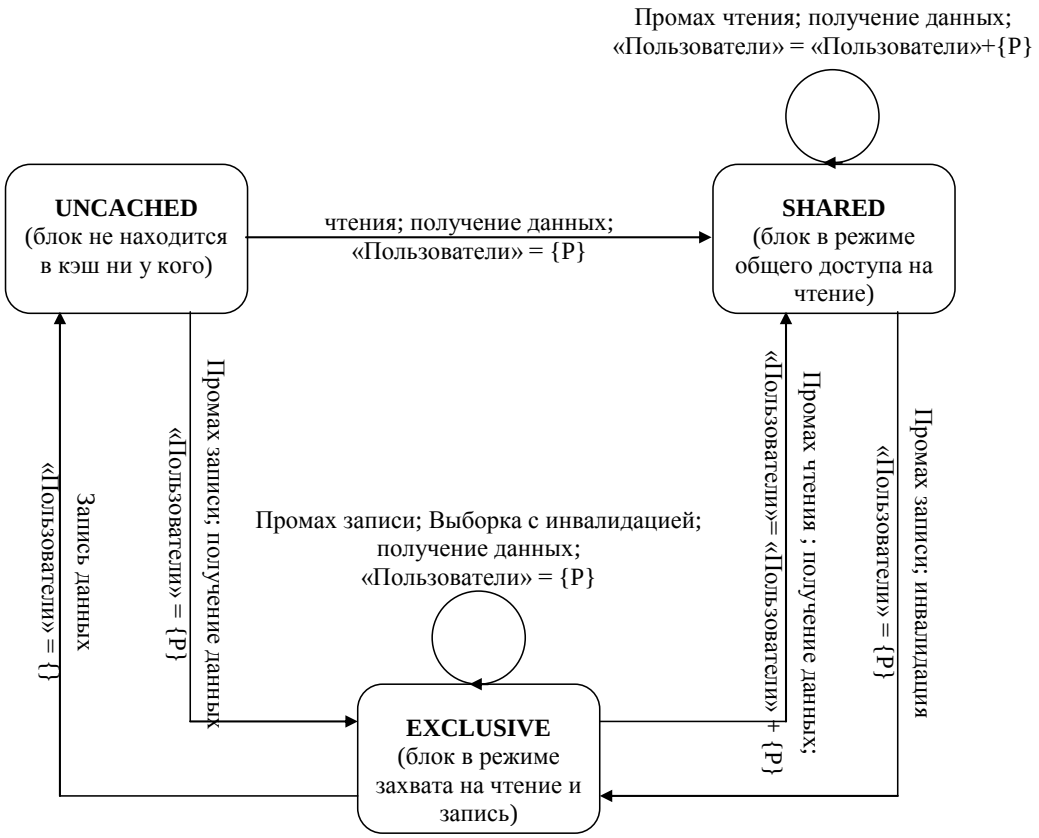
Тип сообщения	**Отправитель**	**Получатель**	**Действия**
Read Miss - промах чтения процессора P блока A	Локальный узел процессора P	Узел R с записью словаря для A	Запросить данные блока A для P; Прописать в словаре у R статус для A «общий доступ» с процессором P
Writ Miss - Промах записи процессора P блока A	Локальный узел процессора P	Узел R с записью словаря для A	Запросить данные блока A для P; Прописать в словаре у R статус для A «эксклюзивный захват» процессором P
Invalidate Инвалидация блока A в узле владельце Q	Узел R с записью словаря для A	Узел Q, хранящий блок A	Проставить признак инвалидации блока A, который был в разделяемом доступе у узла владельца Q
Fetch Выборка A у узла владельца Q	Узел R с записью словаря для A	Узел Q, хранящий блок A	Выбрать содержимое D блока A и переслать актуальное содержимое из узла Q в узел R с его записью словаря
Fetch/Invalidate Выборка с инвалидацией A у узла владельца Q	Узел R с записью словаря для A	Узел Q, хранящий блок A	Выбрать содержимое D блока A, проставить ему статус не валидного и переслать актуальное содержимое из узла Q в узел R с

			его записью словаря
Data reply - Получение данных D блока A	Узел R с записью словаря для A	Локальный узел процессора P	Переслать данные D блока A запросившему процессору P из R
Data write back -Запись данных D блока A	Узел Q, хранящий блок A	Узел R с записью словаря для A	Записать данные D блока A в оперативную память процессора Q владельца A

В терминах приведенных в таблице сообщений можно описать диаграмму состояний блока памяти A, которую реализует узел-владелец Q.



Аналогичную схему для отслеживания состояния записи словаря реализует узел R, владелец записи словаря для блока памяти A.



Следует отметить, что поскольку все события инициируются извне, и сам владелец записи словаря не инициирует никаких событий, кроме обновления статуса записи в словаре, то в диаграмме переходов отсутствуют атрибуты «ОТ» и «В». Кроме того, в словаре поддерживается список текущих процессоров-пользователей соответствующего блока, обозначенного на диаграмме как «Пользователи». Очевидно, что у не закэшированного блока нет пользователей, если блок находится в эксклюзивном состоянии, то процессор-пользователь только один, в состоянии общего доступа пользователей может быть много. Аналогично системам, применяющим подход с *отслеживанием,* при использовании *словарей* ни процессоры-пользователи блоков памяти, ни процессоры-владельцы блоков памяти, ни процессоры-владельцы записей словаря не могут «захватить» блок памяти в каком-либо из состояний. Таким образом, запретить изменение состояния блоков памяти не может никто. Оно происходит сразу по запросу от других процессоров, а словарь лишь обеспечивает атомарность операций и играет роль синхронизатора доступа, аналогично шине в подходе с отслеживанием.