КОГЕРЕТНОСТЬ КЭША В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ

ЧЕРВЯКОВ ГРИГОРИЙ

СТ М3235, Itmo University, Russia, St. Petersburg

Abstract. Когда мы переходим от однопроцессорной к многопроцессорной системы возникает множество проблем, которые необходимо решать для построения качественной архитектуры и одна из них это когерентность(согласовонность) кэша. Существует множество подходов к решению этой проблемы. Мы рассмотрим два механизма когерентности кэша, следовательно два способа организации памяти(snooping и directory-based).

1. Coherency

1.1. **Некоторые предостережения.**

Стоит понимать, что существуют модели памяти с общим кэшем. Преимущество такой организации заключается в том, что поддержка когерентности кэша не требуется, поскольку только один кэш, и любой блок может быть кэширован только в одном месте. Недостатки такой организации заключаются в том, что процессоры и кэши должны находиться в непосредственной близости друг от друга, иначе время доступа к кэшу будет очень велико. Кроме того, при наличии нескольких банков кэш-памяти связь между процессорами и банками кэш-памяти осуществляется по принципу all-to-all, что требует больших затрат, поэтому такая организация не является масштабируемой.

Также стоит понимать, что проблемы incoherence происходят из того что существует множество акторов с доступом к кэшу и памяти. В современных системах такими участниками являются процессорные ядра, механизмы DMA и внешние устройства, которые могут читать и/или писать в кэш и память. Мы сфокусируемся на акторах, которые являются ядрами. Мы будем рассматривать упрощенную структуру памяти в которой у нас будет одноуровневый кэш и основная память.

1.2. **Проблема несогласованности.**

Чтобы представить проблему несогласованности, сначала представим систему с несколькими процессорами(fig 1).

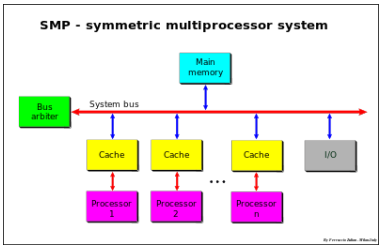

FIGURE 1. SMP – симметричная многопроцессорная система

Рассмотрим следующий код, в котором два потока выполняются на двух процессорах, прибавляя значения `a[0]` и `a[1]` к переменной `sum`

```
int sum = 0;
#pragma omp parallel for
for (int i = 0; i < 2; i++){
  #pragma omp critical {
    sum = sum + a[i];
  }
}
print sum;
```

Предположим, что `a[0] = 3`, а `a[1] = 7`. Правильным результатом вычисления будет то, что сумма отражает сложение `a[0]` и `a[1]` в конце вычисления, что соответствует значению 10. В системе без кэша Tread 0 считывает из памяти начальное значение суммы, прибавляет к нему значение 3 и сохраняет его обратно в память. Tread 1 считывает из памяти значение `sum` (которое теперь равно 3), прибавляет к нему 7 и возвращает в память значение 10. Когда Tread 0 считывает `sum` из памяти, он считывает 10 и печатает его, что правильно.

Теперь представьте, что у каждого процессора есть кэш (write-back) (fig 1). Предположим, что изначально значение суммы в памяти равно 0. Tread 0 начинается с чтения из ячейки памяти, в которой находится переменная `sum`, в регистр. В результате блок памяти, содержащий `sum`, кэшируется Proc 0. Затем Tread 0 выполняет `sum + a[0]`. Результат сложения, который в данный момент все еще находится в регистре, записывается в ячейку памяти `sum`. Поскольку блок, содержащий сумму, кэшируется, блок кэша модифицируется и устанавливается бит dirty. В это время в основной памяти все еще хранится устаревшая копия блока, в которой значение суммы равно 0. Когда Tread 1 выполняет чтение суммы из основной памяти, он видит значение суммы равным 0. Он складывает `a[1]` и сумму и сохраняет результат в кэше. Значение `sum` в его кэш-копии равно 7. Наконец, когда Tread 0 печатает значение `sum`, он считывает его из ячейки памяти `sum`, и он считывает его из кэша, потому что в его кэше есть действительная копия. Он выведет “3”, что неверно. Нетрудно понять, что если кэш будет write-through, то проблема останется. Просто в памяти и в кэше Tread 1 у нас будут корректные значения `sum`.

Как мы видим, проблема не зависит от того как мы проталкиваем значение в основную память, ведь значения в кэшах разных процессоров остаются несогласованные. Это мы и называем **cache coherence problem**.

Существует два критерия, которые вынуждают согласованность кэшей:

1. **Write propogation** - Изменения данных в любом кэше должны быть распространены на другие копии (этой строки кэша) в соседних кэшах.
2. **Transaction serialization** - Чтение/запись в одну ячейку памяти должны восприниматься всеми процессорами в одном и том же порядке.

Существует несколько эквивалентных определений этого инварианта существования согласованности. Мы будем в основном пользоваться этим. Также мы вводим понятие coherence protocol, который будет следить за соблюдением инварианта.

1.3. **Coherence protocol in general.**

В этой части мы постараемся обобщить все виды протоколов когерентности. Чтобы реализовать эти инварианты на практике, мы "присоединяем" **finite state machine (FSM)**, определяемую как контроллер когерентности, к каждой структуре хранения, включая память и каждый кэш(блок у процессора). Эти контроллеры когерентности собираются вместе, образуя распределенную систему. Контроллеры взаимодействуют друг с другом, чтобы гарантировать, что инварианты всегда сохраняются для каждого блока. Протокол когерентности определяет, как эти конечные автоматы взаимодействуют друг с другом.

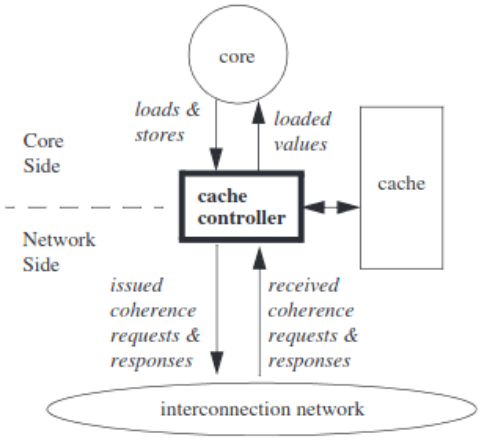

FIGURE 2. Coherency controller(cache controller)

Каждый контроллер когерентности управляет набором конечных автоматов, которые концептуально являются одним и тем же, но независимым автоматом для каждого блока. В зависимости от текущего состояния блока эти машины получают и обрабатывают события, такие как входящие сообщения когерентности. Контроллер когерентности реагирует на событие типа A (например, запрос на сохранение от ядра к контроллеру кэша) в блоке B, выполняя действия (например, запрашивая разрешение на чтение и запись), которые зависят как от A, так и от состояния B (например, только чтение). После выполнения этих действий контроллер имеет возможность изменить состояние B.

Вот так, например, будет выглядеть **FMS** для самого простой протокола когерентности кэша, построенный на основе write-through и используемый **snoopy mechanism** для взаимодействия с остальными котроллерами.(fig 3)

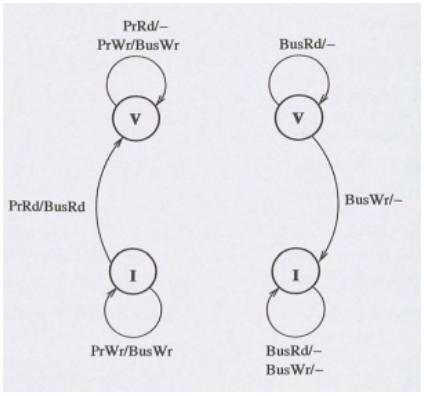

FIGURE 3. FMS

Где запросы процессора к контроллеру:

1. PrRd - прочитать блок.
2. PrWr - записать блок.

Внешние запросы к контроллеру:

1. BusRd - Существует запрос на чтение блока от какого-то из процессоров.
2. BusWr - Существует запрос на запись блока от какого-то из процессоров.(В нашем случае также и запись в память)

Состояния блоков кэша:

1. V(Valid) - Значение блока такое же как и в основной памяти.
2. I(Invalid) - Обращение к этому блоку произведет cache miss.

Чтобы определить coherence protocol необходимо определить 3(Иногда вводят дополнительную кмпоненту **actions**) компоненты.

- States: Состояния блоков кэша и памяти(V, I).
- Events: Запросы, приходящие к контроллеру извне(PrRd, PrWr, BusRd, BusWr).
- Transitions: Это соответствие {State, Event} -> {State}

States относительно независимы, например два класса coherence protocols: snoopy и directory можно описать с помощью одного набора states.

1.4. **States.**

1.4.1. *Types states.*

В одноакторной системе блок кэша может находиться в одном из двух состояний: валидном или невалидном. Если необходимо различать Dirty блоки, блок кэша может иметь два действительных состояния. Dirty блоком считается тот, который был записан позже, чем любая другая копия того же блока.

Эти два или три состояния также могут использоваться системой с несколькими акторами, однако обычно мы хотим различать разные типы действительных состояний. Мы хотим закодировать четыре свойства блока кэша.

- Validity: Можно прочитать, но нельзя писать.
- Dirtiness: Блок кэша считается грязным, если его значение является самым актуальным значением, это значение отличается от значения в памяти, а контроллер кэша отвечает за обновление памяти новым значением.
- Exclusivity: Если это единственная частная кэшированная копия этого блока в системе
- Ownership: Контроллер кэша (или контроллер памяти) является владельцем блока, если он отвечает за ответы на запросы когерентности для этого блока

1.4.2. *Stable states.*

Основные состояния можно определить буквами MSI(иногда используются O и E, но необязательно). Каждое такое состояние это комбинация типов состояний описанных выше.

- Modified: Блок valid, exclusive, owned и potentially dirty. Вы можете либо прочитать, либо записать блок. Копия блока в памяти может быть устаревшей, в кэше хранится единственная действительная копия блока, и кэш должен отвечать на запросы к блоку.
- Shared: Блок valid, но не exclusive, не dirty и не owned. В кэше хранится копия блока, доступная только для чтения. В других кэшах могут быть валидные копии блока, доступные только для чтения.
- Invalid: Блок либо отсутствует в кэше, либо он там есть, но может быть устаревшей копией, которую кэш не может прочитать или записать.
- Owned: Блок valid, owned, and potentially dirty, но не exclusive. Кэш должен отвечать на запросы к блоку и содержит его копию, доступную только для чтения. Копия блока, доступная только для чтения, может существовать в других кэшах, но им она не принадлежит. Возможно, копия блока в памяти устарела.
- Exclusive: Блок valid, exclusive, and clean. В кэше хранится копия блока, доступная только для чтения. Копия блока в памяти является самой последней копией, и ни в каких других кэшах нет действительных копий блока. Хотя существуют протоколы, в которых состояние Exclusive не рассматривается как состояние владения, для целей данного учебника мы будем считать, что блок является владельцем, когда он находится в состоянии Exclusive.

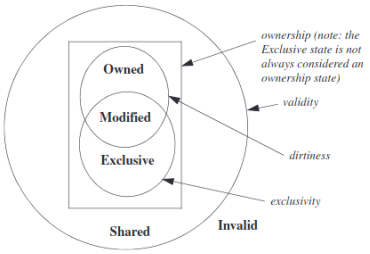

FIGURE 4. MOESI states

1.5. **Snooping and Directory-based.**

Сейчас дадим кртакое описание каждого из механизмов, а в дальнейшем углубимся в каждый по отдельности.

- **Snooping**: Контроллер кэша передает сообщение запроса всем другим контроллерам когерентности, чтобы начать запрос блока. Вместе контроллеры когерентности, например, отвечают на запрос данных от другого ядра, если они являются его владельцем. Способность сети межсоединений посылать широковещательные сигналы каждому ядру в регулярном порядке необходима для протоколов snooping. Большинство протоколов snooping работают в предположении, что запросы поступают в определенном порядке, например по общей проводной шине, однако возможны и более сложные сети межсоединений и гибкие последовательности.
- **Directory-based**: Контроллер кэша отправляет запрос контроллеру памяти. Каждый контроллер памяти имеет каталог, содержащий информацию о каждом блоке памяти, включая данные о текущих владельцах. Контроллер памяти определяет состояние каталога блока при получении запроса на него из дома. Например, если запрос GetS, контроллер памяти просматривает состояние каталога, чтобы определить владельца. Если память является владельцем, контроллер памяти завершает транзакцию. Если контроллер кэша является владельцем, контроллер памяти контроллер памяти направляет запрос в кэш владельца; когда кэш владельца повторно получает переданный запрос, он завершает транзакцию, отправляя ответ на данные запрашивающему.

При выборе между Snooping и Directory приходится идти на компромиссы. Хотя методы Snooping логически просты, широковещание не масштабируется до большого числа ядер. Благодаря своей одноадресной природе протоколы Directory масштабируются; тем не менее, поскольку они требуют отправки дополнительного сообщения в некоторых случаях, многие транзакции выполняются медленнее.

1.6. **Invalidate and Update.**

Еще один вопрос, который не зависит от механизма когерентности. Существуют два способа решения:

- Invalidate protocol: Ядро запускает транзакцию когерентности, чтобы убедиться, что копии во всех других кэшах недействительны, когда оно хочет записать в блок. Запрашивающий может записывать в блок, не беспокоясь о том, что другое ядро получит доступ к его предыдущему значению после того, как копии станут недействительными. Для того чтобы другое ядро получило блок после того, как его копия была признана недействительной, должна быть запущена новая транзакция когерентности. При этом оно получит копию от ядра, которое первоначально записало блок, поддерживая согласованность.
- Update protocol: Чтобы обновить копии в других кэшах новым значением, присвоенным блоку, ядро, которое хочет записать блок, запускает транзакцию когерентности.

И тут нам опять приходится выбирать между скоростью при чтении(Update protocol) и скорости при записи(Invalidate protocol).

Дальше мы рассмотрим несколько примеров реализации coherence protocols

2. Snooping Coherence Protocols

2.1. **Write-Once protocol.**

Первый MESI протокол

2.1.1. *States.*

- Invalid: Некорректное значение в кэше
- Valid: Значение когерентно. Shared, но не Modified
- Reserved: Блок является единственной копией памяти, но он по-прежнему когерентен. При замене блока не требуется обратная запись.
- Dirty: Блок - это единственная копия памяти, и она некогерентна. Эта копия была записана один или несколько раз. Это единственное состояние, которое генерирует обратную запись при замене блока в кэше.

2.1.2. *Transitions.*

- Read hit: Информация актуальная. никаких изменений
- Read miss: Если существует другая копия блока, находящаяся в состоянии Dirty, кэш с этой копией запрещает памяти поставлять данные и поставляет сам блок, а также записывает блок обратно в основную память. Если ни один кэш не имеет копии Dirty, блок поступает из памяти. Все кэши с копией блока устанавливают свое состояние в Valid.
- Write hit: Если блок уже находится в состоянии Dirty, запись может быть выполнена локально без задержки. Если блок находится в состоянии Reserved, запись также может быть выполнена без

задержки, а состояние меняется на Dirty. Если блок находится в состоянии Valid, записываемое слово записывается в основную память (т. е. получает шину, и происходит запись одного слова в резервное хранилище), а локальное состояние устанавливается в Reserved. Другие кэши с копией этого черного цвета (если таковые имеются) наблюдают за записью по шине и изменяют состояние своих копий блока на Invalid. Если блок заменяется в состоянии Reserved, его не нужно записывать обратно, поскольку копия в основной памяти является текущей.

- Write miss: В случае частичной записи строки кэша за пропуском чтения (если это необходимо для получения незаписанной части строки кэша) следует пропуск записи. В результате текущий кэш находится в состоянии Reserved, а все остальные кэши - в состоянии Invalid.

2.2. **Synapse protocol.**

2.2.1. *States.*

Invalid, Valid, Dirty.

2.2.2. *Transitions.*

- Read hit: Данные берутся из текущего кэша. В состоянии ничего не меняется.
- Read miss: Кэш, отправивший Read Miss, получает отрицательное подтверждение, если в другом кэше есть Dirty-копия. После этого владелец записывает блок обратно в основную память, одновременно устанавливая локальное состояние Invalid и сбрасывая битовую метку. Чтобы получить блок из основной памяти, запрашивающий кэш должен послать еще один запрос на пропуск. Во всех остальных случаях блок забирается прямо из основной памяти. Владелец блока, будь то память или кэш, всегда поставляет его. Загруженный блок постоянно находится в состоянии Valid.
- Write hit: Запись может начаться сразу, если блок Dirty. Поскольку сигнала о недействительности нет, если блок Valid, процесс будет таким же, как и при Write Miss (включая полную передачу данных).
- Write miss: Блок всегда берется из памяти, как и пропуск чтения; если блок был Dirty в другом кэше, владелец должен сначала записать его в память. Блок загружается в состоянии Dirty, и все кэши, содержащие копию Valid-блока, устанавливают свой статус в Invalid. Метка основной памяти для блока устанавливается так, чтобы игнорировать любые дальнейшие запросы к блоку из памяти.

2.3. **Dragon protocol.**

2.3.1. *States.*

- Valid-Exclusive: Блок кэша был захвачен этим процессором и не изменен другим.

- Shared clean: Это указывает на то, что блок кэшируется более чем одним процессором, что доказывает, что текущий процессор не является последним, кто его записывал.

Протокол поддерживает состояния E и Sc раздельно, чтобы избежать операций чтения-записи в блоках кэша без общего доступа, вызывающих транзакции по шине, которые замедляют выполнение. В однопоточных приложениях такое часто случается.

- Shared dirty: Это означает, что блок присутствует в кэшах нескольких процессоров, причем текущий процессор является последним, кто его изменил. В результате процессор, используемый в данный момент, называется владельцем блока. В отличие от протоколов invalidation, блок должен быть только в процессоре, а не в основной памяти. Когда блок кэша удаляется, процессор обязан обновить основную память.
- Dirty: Только этот процесс имеет его в своей кэш-памяти и к тому же он его изменил.
- Sharedline - также необходима для указания того, доступен ли определенный блок кэша в нескольких кэшах.

2.3.2. *Transitions.*

- Read Miss - Если в другом кэше есть копия Dirty или Shared-dirty, он предоставит блок кэша, поднимет SharedLine и состояние блока будет установлено в shared-dirty во всех кэшах. Если в другом кэше есть блок в состоянии Valid-Exclusive или shared-clean, он предоставит блок, поднимет SharedLine и состояние блока будет установлено в shared-clean во всех кэшах. Если SharedLine не был поднят, блок будет получен из памяти с состоянием Valid-Exclusive.
- Write Hit - Если блок находится в состоянии Dirty или Valid-Exclusive, запись может быть выполнена немедленно, с конечным состоянием Dirty. Если блок находился в состоянии shared-clean или shared-dirty, запись задерживается до тех пор, пока не будет получена шина и не начнется запись в основную память. Другие кэши наблюдают за записью на шину и обновляют свою копию блока.
- Write Miss - Как и при пропуске чтения, блок берется из кэша, если он Dirty или Dirty, и из памяти в противном случае. Другие кэши с копиями устанавливают свое локальное состояние в shared-clean. При загрузке блока запрашивающий кэш устанавливает локальное состояние в Dirty, если SharedLine не поднят. Если SharedLine поднят, запрашивающий кэш устанавливает состояние shared-dirty и выполняет запись на шину для трансляции нового содержимого.

3. Directory-based

3.1. **States.**

- Shared (S) - Один(или больше) процессор имеет блок у себя в кэше при этом согласован с памятью и с другимим кэшами.
- Uncached (U) - Некто не кэшировал этот блок
- Modified (M) - Только один процессор кэшировал этот блок. Также этот процесс писал в него, следовательно этот блок не согласован с памятью

3.2. **Transition.**

Initial state	Request	Response/ Action	New state
U	Read Miss или Write Miss	Возьми блок прямо из памяти. Отправь блок из памяти тому, кто его запросил.	M
M	Read Miss	Отправьте запрос в кэш, содержащий измененный блок, на предоставление данных в запрашивающий кэш	S
	Write Miss	Отправьте запрос в кэш, содержащий измененный блок, чтобы аннулировать его	-
S	Read Miss	Ответьте запрашивающему кэшу блок памяти	-
	Write Miss	Ответьте запрашивающему кэшу блок памяти. Отправьте запрос всем кэшам, которые совместно используют этот блок, чтобы аннулировать его.	M
	Write Hit	Отправьте запрос всем кэшам, которые совместно используют этот блок, чтобы аннулировать его. Ответьте кэшу, что блок может быть модифицирован.	M

3.3. **Implementations.**

3.3.1. *Full bit vector format.* Этот метод хранит битовое поле для каждого процессора в узле каталога. Количество процессоров определяет, насколько велики накладные расходы на хранение.

FIGURE 5. Full bit vector format

3.3.2. *Limited pointer format.* Чтобы сократить накладные расходы на хранение, этот метод хранит информацию о каталоге для ограниченного числа блоков в каталоге.

3.3.3. *Number-balanced binary tree format.* В этом формате каталог децентрализован и распределен между кэшами, разделяющими блок памяти. Различные кэши, разделяющие блок памяти, располагаются в виде двоичного дерева. Кэш, который первым обращается к блоку памяти, является корневым узлом.

References

1. Yan, S.: Fundamentals of parallel multicore architecture. (2010)
2. https://arxiv.org/pdf/cs/0208027.pdf
3. Hennessy, J.: Computer Organization and Design.
4. A Primer on Memory Consistency and Cache Coherence.
5. http://www.cs.ucr.edu/~ravi/Papers/NWConf/ravishankar_83.pdf
6. The Cache Memory Book.