1 Что такое когерентность кэша?

1.1 Определение и назначение когеренетности кэша

Когерентность кэша (англ. *cache coherence*) - свойство кэшей, означающее целостность данных, хранящихся в локальном кэшах для разделяемого ресурса. Очевидно, что если у нас одни и те же данные могут быть скэшированы в кэш-памяти разных уровней, и в локальных блоках кэш-памяти разных процессоров/ядер – при модификации этих данных легко может случиться, что существует множество несогласованных версий одних и тех же данных. Можно, конечно, возложить обязанность разруливать все эти конфликты на программиста – но это слишком жестоко. Поэтому производители процессоров обычно реализуют какой-либо аппаратный механизм поддержания согласованности данных различных кэшей между собой, и с основной памятью.
Возможных вариантов организации когерентности – пруд пруди. Поскольку меня интересуют многоядерные системы – я буду рассматривать наиболее часто используемую здесь реализацию когерентности через "подслушивание"(англ. *snooping*).

1.2 Что такое *snooping*?

У нас есть общая шина, к которой подключены кэши всех ядер, и к ней же подключен контроллер основной памяти. Любые запросы отправляются бродкастом на шину, и видны всем участникам вечеринки сразу. Принципы коммуникации через шину:

1. Шина считается надежной средой – т.е. нет никаких циклов запрос-подтверждение. Если я запустил по шине какую-то транзакцию, и она завершилась – значит все участники ее видели, и все с ней согласны. Не возможна ситуация, что кто-то мог не получить сообщение, или кто-то мог не успеть ответить. Разумеется, это все с точки зрения высокоуровневых протоколов – на уровне реализации в железе, возможно, такие циклы и есть.

2. Если я хочу что-то сделать, я не спрашиваю остальных участников «согласны ли вы?» – я просто начинаю транзакцию, реализующую мое намерение. Начатую мной транзакцию видят все участники, поэтому если кто-то из них имеет что-то сказать против – он прерывает транзакцию, и начинает свою. Влезать в чужую транзакцию – это вполне регулярный элемент протоколов кэш-когерентности.

3. При этом все участники действуют все-таки кооперативно – например, «протестующий» участник не тянет тупо одеяло на себя, а помогает исходной транзакции, которую он прервал – просто выполняя свою дополнительную транзакцию, он в чем-то подправляет исходную. Другой пример: если я прервал чью-то транзакцию, все участники коммуникации полагают, что мне есть что сказать важного, поэтому, на ближайший такт все (кроме меня) добровольно замолкают, давая мне возможность спокойно высказаться.

2 Протоколы когерентности

2.1 MSI

MSI является представителем класса простейших протоколов когерентности кэша. В MSI каждый блок, содержащийся в кэше может иметь одно из следующих состояний:

- **M**odified: данный блок был изменен в кэше и не совпадает со своим аналогом в памяти. Если кэш-линия под флагом *M* у одного из ядер, она может встречаться у остальныъ только в состоянии *I*.
- **S**hared: данный блок не модифицирован и присутствует в *read-only* состоянии хотя бы в одном кэше. Кэш может вытеснять данные, не записывая их в память.
- **I**nvalid: данный блок либо не размещен в текущем кэше, либо был инвалидирован по запросу шины. Такие линии - первые в очереди на удаление.

Для любой пары кэшей допустимые состояние данной кэш-линии следующие:

	M	S	I
M	✗	✗	✓
S	✗	✓	✓
I	✓	✓	✓

Эти состояния когерентности поддерживаются посредством обмена данными между кэшами и памятью. Кэши имеют разные обязанности, когда блоки читаются или записываются, или когда они узнают о других кэшах, выдающих чтение или запись для блока.

Как работает MSI?

Когда запрос чтения поступает в кэш для блока в состояниях «M» или «S», кеш передает данные. Если блок не находится в кеше (в состоянии «I»), он должен убедиться, что строка не находится в состоянии «M» в любом другом кеше. Различные архитектуры кэширования обрабатывают это по-разному. В случае snooping архитектуры шины запрос на чтение передается во все кэши. Если в другом кеше есть блок в состоянии «M», он должен записать данные в память и перейти к состояниям «S» или «I». После того, как любая строка «M» будет записана обратно, кеш получит блок из памяти или другого кеша с данными в состоянии «S». Затем кеш может передать данные запрашивающему. После подачи данных блок кэша находится в состоянии «S».

Когда запрос на запись поступает в кэш для блока в состоянии «M», кеш изменяет данные локально. Если блок находится в состоянии «S», кэш должен уведомлять другие кеши, которые могут содержать блок в состоянии «S», чтобы они инвалидировали блок. Затем данные могут быть локально изменены. Если блок находится в состоянии

«I», кэш должен уведомить другие кэши, которые могут содержать блок в «S» или «M» состояние, что они должны инвалидировать блок. Если блок находится в другом кэше в состоянии «M», этот кеш должен либо записывать данные в память, либо передавать их в запрашивающий кеш. Если в этот момент кэш еще не имеет блока локально, блок считывается из памяти до его изменения в кеше. После изменения данных блок кэша находится в состоянии «M».

Проблемы MSI

- Многочисленные обращения к памяти
- Необходимость сообщать о модификации S, в то время как обычно с каждой кеш-строкой в конкретный момент времени работает лишь одно ядро

MSI не так плоха как кажется, ведь большинство обращений к памяти – это обращения к L3, который общий для всех ядер. Тем не менее, на данный момент MSI совсем не используется на практике.

2.2 MESI

MESI является одним из наиболее распространенных протоколов, поддерживающий *write-back* кэши. Данный протокол уменьшает нагрузку на шину, что было неотъемлимым недостатком MSI. Рассмотрим дополнительное состояние в этом протоколе:

- **E**xclusive: данный блок присутствует только в текущем кэше и является «чистым». Он может быть изменен на состояние *Shared* в любое время в ответ на запрос на чтение. Также он может быть изменен на *Modified* состояние при записи на него.

Для любой пары кэшей допустимые состояние данной кэш-линии следующие:

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓

Можно заметить, что количество бит для хранения состояния кэш-линии не увеличилось по сравнению с предыдущим протоколом.

Как работает MESI?

Если в какой-то момент кэш посылает запрос на чтение кэш-линии, которой у него нет, другие кэши, которые имеют эту строку в состоянии «S»/«E» отвечают, что у них есть

эта кэш-линия. Если таких сообщений не поступило, то присваиваем новой кэш-строке статус «E». Также «E» следует менять на «S», когда “наше” ядро сообщает о наличии кэш-линии другим ядрам(ведь в этот момент они читают нашу кэш-строку и она перестает быть Exclusive). Таким образом, теперь очень часто мы сможем переводить из состояния «E» в «M» без каких-либо сообщений по шине.

2.3 MESIF

MESI помогал снизить трафик на шине. Но мы хотим еще снизить количество запросов на общую память (по факту - RAM + L3).
Intel усовершенствовало MESI добавив следующее состояние, которое является специализацией предыдущего «S»:

- **F**orward: значит, что данный кэш является назначенным ответчиком на любой запрос данной кэш-линии. Протокол гарантирует, что если какой-то кэш хранит в себе строку в состоянии «S», то не более одного кэша хранят ее в состоянии «F».

Для любой пары кэшей допустимые состояние данной кэш-линии следующие:

	M	E	S	I	F
M	✗	✗	✗	✓	✗
E	✗	✗	✗	✓	✗
S	✗	✗	✓	✓	✓
I	✓	✓	✓	✓	✓
F	✗	✗	✓	✓	✗

Как работает MESIF?

В системе кэшей, использующей протокол MESI, запрос кэш линии, который принимается несколькими кэшами, содержащими линию в состоянии «S», будет обслуживаться неэффективно. Он может быть либо удовлетворен (медленной) основной памятью, либо все кэши обмена могут отвечать, бомбардируя запросчика избыточными ответами. В системе кэшей, использующей протокол MESIF, запрос строки кэша будет отвечать только кэшем, содержащим строку в состоянии «F». Это позволяет запрашивающему получить копию со скоростью кеш-кэш-памяти, позволяя при этом использовать несколько пакетов многоадресной рассылки, которые позволит сетевая топология. Поскольку кеш может в одностороннем порядке отбрасывать (аннулировать) строку в состояниях «S» или «F», возможно, что в кеше нет копии в состоянии «F», даже если копии в состоянии S существуют. В этом случае запрос на строку выполняется (менее эффективно, но все же правильно) из основной памяти. Чтобы свести к минимуму вероятность отклонения строки F из-за отсутствия интереса, последнему запросчику строки присваивается состояние «F»; когда кэш в состоянии F отвечает, он возвра-

щает состояние F новому кэшу. Мы не только снижаем вероятность удаления этой кэш-линии, то и уменьшаем наггрузку на кэши (не один кэш отправляет свою строку). Таким образом, основное отличие от протокола MESI заключается в том, что запрос на копирование строки кэша для чтения всегда входит в кеш в состоянии F. Единственный способ войти в состояние S - удовлетворить запрос на чтение из основной памяти.

2.4 MOESI

Рассмотрим еще одно расширение MESI от AMD. Добавим следующее 5-е состояние и рассмотрим как изменился Shared:

- **O**wned: Этот кеш является одним из нескольких с допустимой копией строки кэша, но имеет исключительное право вносить в него изменения. Он должен транслировать эти изменения во всех других кешах, разделяющих эту строку. Введение Owned состояния позволяет осуществлять грязное совместное использование данных, то есть модифицированный блок кеша может перемещаться по различным кэшам без обновления основной памяти. Линия кэша может быть изменена в «M» состояние после аннулирования всех общих копий или изменена в состояние Shared, записав изменения в основную память. Owned строки кэша должны отвечать на запрос snoop данными.

- **S**hared: Эта строка является одной из нескольких копий в системе. У этого кеша нет разрешения на изменение копии. Другие процессоры в системе могут также хранить копии данных в Shared состоянии. В отличие от протокола MESI, общая кэш-строка может быть грязной по отношению к памяти; если это так, в каком-то кеше есть копия в состоянии Owned, и этот кеш отвечает за окончательное обновление основной памяти. Если кеш не удерживает строку в состоянии Owned, копия памяти обновляется. Строка кэша не может быть записана, но может быть изменена в «E» или «M» состояние после аннулирования всех общих копий. (Если бы кэш-строка принадлежала ранее, то недействительный ответ будет указывать на это, и состояние станет «M», поэтому обязательство в конечном итоге записать данные обратно в память не забывается.) Оно также может быть отброшено (изменено на Invalid состояние) в любое время. Shared линии кэша могут не отвечать на запрос snoop с данными.

Для любой пары кэшей допустимые состояние данной кэш-линии следующие:

	M	O	E	S	I
M	✗	✗	✗	✗	✓
O	✗	✗	✗	✓	✓
E	✗	✗	✗	✗	✓
S	✗	✓	✗	✓	✓
I	✓	✓	✓	✓	✓

Хотя MOESI может быстро делиться грязными линиями кеша из кеша, он не может быстро обмениваться чистыми линиями из кеша. Если строка кэша чиста по отношению к памяти и в общем состоянии, тогда любой запрос snoop к этой строке кэша будет заполняться из памяти, а не из кэша.

Если процессор хочет записать строку Owned cache, он должен уведомить других процессоров, которые используют эту строку кэша. В зависимости от реализации он может просто сказать им сделать недействительными свои копии (переместить свою собственную копию в состояние Modified), или он может сказать им обновить свои копии новым содержимым (оставив свою собственную копию в состоянии Owned).

FAQ

- Зачем нам нужна когерентность?
- Когда начинает требоваться когерентность?
- Можно ли обойтись без когерентности?
- Поподробнее про ошибки когерентности из двух кэшей?
- Ну и как их исправить без когерентности?
  Ответ: вынести кэши за шину, но это медленно работает
- В каких случаях когерентность не возникает при трех кэшах?
  Ответ: если данные ядер не пересеаются
- Чем отличаются M и O?
- MOESIF?
  Ответ: еще никто не сконструировал MOESIF
- Как получить O при пустом кэше (изначально пустой/чтобы был пустой, т.е. Invalid остальные)?
- Как получить F (изначально пустой/чтобы был пустой, т.е. Invalid остальные)?