1 Зачем оно надо?

Представим, что у нас есть два процессора CPU0 и CPU1. CPU0 считал строку X. Теперь она есть в CPU0_Cache. CPU1 считал строку X. Теперь она есть в CPU1_Cache. CPU0 пишет в строку X новое значение и возвращает в память. CPU1 снова считывает строку X и получает старое закэшированное значение. Очень жаль.
Чтобы такого не происходило, придумали протоколы и механизмы когерентности.

2 Механизмы когерентности

Два самых известных механизма когерентности это *snooping* и использование *директорий*(directory-based), у каждого из них есть преимущества и недостатки.

2.1 Snooping

Каждый запрос передается (*broadcast*) всем другим узлам.
Протоколы, базирующиеся на этом механизме, быстрее, если у шины достаточная пропускная способность, т.к. все запросы/ответы видны всем процессорам. Недостаток в том, что такие системы плохо масштабируются. Каждый запрос должен передаваться всем узлам сети, а это значит, что вся система становится больше, ширина шины и ее пропускная способность тоже должны увеличиваться.

2.2 Использование директорий*

При таком механизме, общие данные хранятся в специальной общей директории, которая и решает проблему когерентности: при запросе на чтение процессор должен спросить разрешение у директории на то, чтобы загрузить данные из памяти себе в кэш. Если процессор изменяет кэш-линию, то директория либо обновляет данные других кэшей, либо говорит им выкинуть свои копии. Директории медленнее, но не нуждаются в широкой шине, т.к. сообщения передаются от одного узла к другому, а не всем узлам сети. По этой причине многие многопроцессорные системы (>64 CPU) используют именно этот механизм.

3 Протоколы когерентности кэш-памяти

3.1 MSI

3.1.1 Описание

- **M**odified - кэш-линия была изменена, т.е данные в кэше не совпадают с данными в памяти. Кэш несет ответственность за запись такой линии в память.
- **S**hared - кэш-линия не изменен и присутствует в read-only состоянии хотя бы в одном кэше. Кэш может выкинуть такую линию без записи в память.
- **I**nvalid - кэш-линия или не присутствует в данном кэше, или была аннулирована запросом шины.

Запрос на чтение

В CPU0_Cache пришел запрос на чтение кэш-линия X.

- Если X Modified или Shared, CPU0_Cache возвращает кэш-линия X, и всё хорошо.
- Если же X Invalid, то CPU0_Cache должен выяснить, не находится ли эта линия где-то в другом кэше в состоянии Modified.
  - Если в CPUN_Cache линия X находится в состоянии Modified, он должен записать эту кэш-линию в память и перевести её в состояние Shared или Invalid. После этого CPU0_Cache считывает кэш-линию X из памяти или у кэша, у которого блок в состоянии Shared.
  - Если в CPUN_Cache кэш-линия X находится в состоянии Shared, CPU0_Cache получает кэш-линию от CPUN_Cache.
  - Если ни в одном другом кэше нет линии X, кэш обращается к памяти.

После операции чтения кэш-линия X в CPU0_Cache находится в состоянии Shared.
Запрос на запись В CPU0_Cache пришел запрос на запись в кэш-линию X.

- Если кэш-линия в состоянии Modified, кэш изменяет её локально и всё у него хорошо.
- Если кэш-линия в состоянии Shared, то кэш говорит всем остальным выкинуть свои копии. Далее данные могут быть изменены локально.
- Если кэш-линия в состоянии Invalid, то кэш должен сообщить всем остальным кэшам выкинуть свои копии. Если в каком-то из кэшей линия в состоянии Modified, то этот кэш может как записать кэш-линию в память, так и передать её CPU0_Cache. Это зависит от реализации. Если на этот момент CPU0_Cache еще не получил кэш-линию X, он запрашивает её у памяти.

После операции записи кэш-линия X в CPU0_Cache находится в состоянии Modified

	M	S	I
M	✗	✗	✓
S	✗	✓	✓
I	✓	✓	✓

Рис. 1: Разрешенные состояния кэш-линии X

3.1.2 Проблемы?

Да, чаще всего у процессора какие-то свои данные, которые есть только у него. Тем не менее, каждый раз, когда ему надо сделать запись в Shared, он держит в курсе остальных, что надо бы эти данные выкинуть. Шина забивается мусорными сообщениями.

4 MESI

4.1 Что нового?

Добавили еще одно состояние, теперь:

- **S**hared - кэш-линия содержится *больше*, чем в одном кэше.
- **E**xclusive - кэш-линия содержится *только* в одном кэше и не изменена.

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

Рис. 2: Разрешенные состояния кэш-линии X

4.2 Есть проблемы?

Да, когда несколько процессоров пишут в одну и ту же кэш-линию, ее постоянно приходится сбрасывать в память. А еще когда процессор запрашивает линию, которая в нескольких других находится в состоянии Shared, то отвечают все и забивают шину.

5 MOESI

Это вариация AMD на тему MESI.

5.1 Что изменилось?

Добавили еще одно состояние, теперь:

- **O**wned - этот кэш один из нескольких, у кого есть копия этой кэш-линии, но только он может вносить изменения. Состояние Owned позволяет делиться измененными кэш-линиями, не записывая изменения в память. Кэш-линия может быть переведена в состояние Modified, после того, как все остальные кэши выкинут свои копии, и в состояние Shared, после записи изменений в память.

- **S**hared - эта кэш-линия - одна из нескольких копий в системе. Кэш не может вносить изменения в эту кэш-линию. Линия может как совпадать с памятью (ни у одного кэша нет этой линии в состоянии Owned), так и быть измененной (где-то есть кэш, у которого эта кэш-линия Owned). Кэш-линия может быть переведена в состояние Modified или Exclusive, после того, как остальные кэши выкинут свои копии.

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

Рис. 3: Разрешенные состояния кэш-линии X

5.2 Преимущества

Из-за того, что не можем не писать измененные данные в память, получаем выигрыш по скорости, если шина от кэша до процессора сильно жирнее и быстрее, чем от кэша до памяти.

5.3 Недостатки

Не можем быстро читать чистые (неизмененные) линии. Если у каких-то двух кэшей есть линия в состоянии Shared, и она не изменена, то третий скорее получит эту линию от памяти, чем от кэшей.

6 MESIF

Вариация **Intel** на тему MESI

6.1 Что нового?

- **F**orward - специальный вид состояния Shared, показывает, что кэш должен отвечать на запросы шины по этой кэш-линии.

6.2 Преимущества

Не загружаем шину ответами кэшей.

7 MOESIF

Можно сделать. Но никто не сделал.