Анализ когерентности кэш-памятей для повышения эффективности тестирования подсистемы памяти

В.В. Рудометов, В.С. Семенов

Введение

Современные тенденции к наращиванию архитектурной производительности микропроцессоров заставляют разработчиков искать более изощренные способы реализации подсистемы памяти, что, в свою очередь, ведёт не только к усложнению разработки аппаратуры, но и к поиску новых алгоритмов и методов проверки соответствия разрабатываемой аппаратуры утверждённым спецификациям.

В статье описан подход к верификации подсистемы памяти современного микропроцессора. Он основан на проверке согласованности состояний кэш-памятей всех уровней иерархии, разработан в виде дополнительного пакета программ и может использоваться при успешном завершении моделирования тестовой программы. Данный подход позволяет расширить область поиска ошибок путем включения проверок протокола когерентности, тем самым позволяет находить ошибки, поиск которых тестированием достаточно сложен, исходя из своей природы. К тому же достаточно сложно, а иногда и невозможно, произвести прямую проверку некоторых спецификаций, в таких случаях использование этого подхода для проверки спецификаций является эффективным решением.

1. Основная часть

1.1. Оптимизации доступа к памяти

Быстрый рост частоты современных микропроцессоров приводит к росту логического времени доступа к оперативной памяти, то есть увеличивается количество тактов микропроцессора, требуемое для получения информации из памяти. Для решения или, по крайней мере, сглаживания этой проблемы используются как аппаратные, так и программные подходы.

Одним из основных аппаратных методов является использование иерархической памяти. Этот метод заключается в том, что в микропроцессор встраиваются сверхоперативные (кэш) памяти для хранения часто используемой информации, что позволяет значительно снизить количество обращений в основную память. В таком случае требование к пропускной способности и времени доступа шины памяти снижается, но в результате процессоры получают доступ к одной и той же памяти, и, следовательно, в кэш-памятях могут находиться копии как разделяемых, так и локальных данных. Локальные данные – это данные, используемые одним процессором, в то время как разделяемые данные используются несколькими процессорами, по существу обеспечивая обмен между ними. Когда кэшируется элемент локальных данных, их значение переносится в кэш для сохранения среднего времени доступа, а также требуемой полосы пропускания. Поскольку никакой другой процессор не использует эту информацию, этот случай идентичен случаю однопроцессорной конфигурации. Если кэшируются разделяемые данные, то разделяемое значение может находиться в кэш-памятях нескольких процессоров. Кроме

сокращения задержки доступа и требуемой полосы пропускания такое разделение данных способствует также общему сокращению количества обменов. Однако кэширование разделяемых данных вызывает новую проблему: когерентность кэш-памяти.

Программные подходы позволяют компенсировать время доступа в основную память посредством предварительной загрузки данных в регистры процессоров и/или в кэш-памяти верхних уровней иерархии. При этом аппаратура должна контролировать модификацию данных в памяти между предварительной подкачкой и их реальным использованием.

1.2. Протоколы поддержки когерентности

Для обеспечения целостности данных, имеющих несколько копий в разных процессорах и в памятях разных уровней иерархии, разрабатываются спецификации протоколов когерентности. Существуют несколько классов таких протоколов:

1. Протоколы на основе справочника (directory based). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределён по узлам системы).
2. Протоколы наблюдения (snooping). Каждый кэш, который содержит копию данных некоторой части физической памяти, имеет также соответствующую копию служебной информации (биты состояния). Контроллеры кэш-памятей обмениваются между собой служебной информацией о состоянии блока данных.

Задачей протокола когерентности кэш-памяти является координация доступа к разделяемым блокам памяти. Кэш-памяти процессоров обмениваются информацией для того, чтобы определить какой процессор в настоящий момент является собственником данных. Одним из наиболее полных протоколов является протокол MOESI (Modified, Owned, Exclusive, Shared, Invalid) [3], в котором доступ процессора к блоку данных определяется состоянием этого блока и типом запроса. Работа данного протокола описывается поведением конечного автомата с 5 состояниями. На верхнем уровне абстракции проверка правильности работы сводится к моделированию поведения автомата и контролю корректности его состояний и не представляет существенной сложности. Однако, детализация работы протокола в реальном устройстве приводит к значительным усложнениям полученной картины. Так, даже для простейшего протокола типа MSI (однопроцессорная система) имеем 11 состояний (8 из которых переходных), 13 возможных событий и 21 действие [3].

1.3. Построение алгоритма проверки протоколов когерентности

Построение алгоритма проверки рассмотрим на основе классической SMP системы в двухпроцессорном варианте, стоит также заметить, что описываемый метод легко может быть расширен и на 4-х, 8-и и более процессорную конфигурацию из-за основных принципов SMP систем. Также для простоты будем рассматривать 2-х уровневую организацию кэш-памяти: кэш первого (L1) и второго уровня (L2) (Рис. 1).

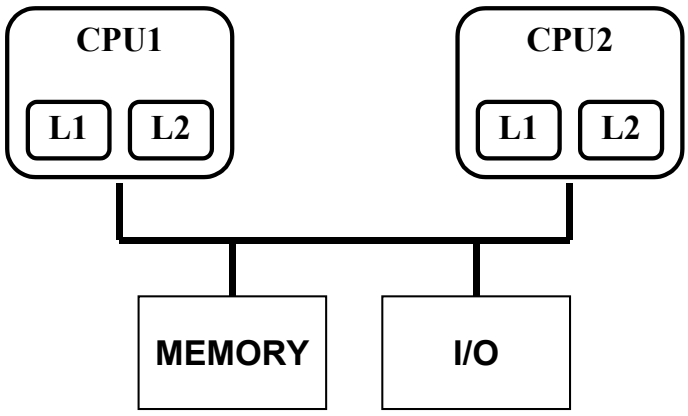

Рис. 1. SMP система с 2-х-уровневой организацией кэш-памятей

Кэш первого уровня (L1) в рассматриваемой системе является множественно-ассоциативным с **NL1** столбцами (**NL1** - way set associative), каждый столбец содержит **IL1** блоков данных (единиц кэша) по **SL1** байт каждый, следовательно, полный размер кэш памяти **NL1xIL1xSL1** байт. L1 обладает сквозной записью (write through, store through), то есть при записи данные пишутся в L1 и в L2. Для каждого блока данных имеются бит значимости VL1 и адресный тэг TL1.

Кэш второго уровня (L2) также является множественно-ассоциативным с **NL2** столбцами, каждый столбец содержит **IL2** блоков данных по **SL2** байт каждый, следовательно, полный размер кэш памяти **NL2xIL2xSL2** байт. Стратегия записи в L2 - запись с обратным копированием (write back, copy back, store in) - модифицированный блок кэш-памяти пишется в основную память только когда он замещается. Для каждого блока данных связаны несколько битов состояния и адресный тэг **TL2**. В биты состояния входят: бит значимости кэш блока **VL2** (valid bit), бит модификации **M** (dirty bit) и бит shared **S**. Следует добавить, что для каждой кэш-памяти столбец и строка однозначно задают адресный тэг и биты состояний.

2. Практическая часть

2.1. Формальные признаки когерентности кэш-памятей

Для построения системы, проверяющей спецификации протокола когерентности, требуется построить формальные правила когерентности, по которым и осуществлять прямые проверки.

Так как в L1 используется сквозная запись, то когерентность для данных из L1 означает следующее, если блок данных значим (**VL1 =1**), то такие же данные должны быть или в оперативной памяти, или в кэш памяти L2 этого же процессора по тому же адресу (исходя из адресного тэга **TL1**):

```
If (VL1(AL1))
        {
                DATAL1(AL1) = DATAL2(AL1) | DATAMEM(AL1);
        }
```

Когерентность для данных из L2 является более сложным понятием, это связано с межпроцессорным взаимодействием в случае многопроцессорной конфигурации, и поэтому определение когерентности для L2 будет включать в себя данные не только из оперативной памяти, но и данные из L2 второго процессора. Итак, можно говорить о когерентности в случаях:

1. блок данных в кэш-памяти любого процессора имеет признаки **VL2** = 1, **M** = 0, **S** = 0, то эти же данные должны быть в оперативной памяти по соответствующему адресу **AL2**:

```
If (VL2(AL2) && !M(AL2) && !S(AL2))
        {
                DATAL2(AL2) = DATAMEM(AL2);
        }
```

2. блок данных в кэш какого-нибудь процессора имеет признаки **VL2** = 1, **S** = 1, то эти же данные могут быть в L2 другого процессора по такому же адресу:

```
If (VL2_cpu1(AL2_cpu1) && !M_cpu1(AL2_cpu1) && S_cpu1(AL2_cpu1))
        {
                DATAL2_cpu1(AL2_cpu1) = DATAL2_cpu2(AL2_cpu1);
        }
```

3. блок данных с признаками: VL2 = 1, M = 1 может присутствовать в кэш-памяти только одного из процессоров.

2.2. Описание реализации метода тестирования

Анализ протоколов когерентности позволяет сделать следующие выводы:

1. Логические ошибки в схемах реализации когерентности данных могут привести к трудно обнаруживаемым при тестировании ошибкам. Причин для этого несколько, во-первых, появление ошибки может отстоять по времени сколько угодно далеко от фактического нарушения протокола когерентности, во-вторых, поскольку данные подкачиваются большими блоками (несколько десятков байт), то велика вероятность того, что испорченные данные вообще не будут использоваться в программе. Большинство ошибок в реализации протокола когерентности данных в многопроцессорной системе может быть обнаружено моделированием случайных тестов, но как показано в [3], некоторые классы ошибок являются трудно обнаруживаемыми и требуют целенаправленного поиска. К таким ошибкам относятся: ис-

кажение типа запроса, потеря запросов/ответов, ситуации, при которых возникают бесконечные ожидания ответов и др. Другим классом трудно обнаруживаемых ошибок является нарушение целостности данных.

2. На основе формальных признаков правильности работы протоколов можно осуществить проверку целостности состояния памяти перед непосредственным завершением моделирования теста.

Предлагаемый подход к тестированию позволяет в значительной мере расширить область проверок в существующих тестах посредством подключения к стандартным проверкам проверку когерентности кэш-памятей. Это достигается подключением к объекту тестирования дополнительного устройства для сброса отладочной информации на диск (сброс дампа) и анализирования непосредственно самой этой информации.

Анализ дампа относится к формальным методам верификации [1] [2], базирующихся на прямой проверке спецификаций, и позволяет обнаружить неверную работу устройства до ее проявления на тестах, а также помогает проверить устройства не видимые простыми программными способами. Например, проверка правильности заполнения памяти тэгов программным путем достигается посредством диагностических запросов в память, но, к сожалению, данный способ не эффективен. Следует заметить, что подключение дампа никак не повлияет на работу процессора в целом, так как не имеет никаких выходных интерфейсов с другими исполнительными устройствами и выполняет только одну функцию - копирование состояния подсистемы памяти на диск.

Сброс дампа осуществляется по окончанию моделирования тестовой программы. В описываемой работе сбрасывание данных осуществлялось только при успешном проходе теста, хотя существует возможность сброса во время прохода теста, но из-за динамических особенностей архитектуры это трудно реализуемо. Поэтому использовалась трассировка обращений к кэш-памятям для облегчения анализа и локализации ошибки. Следует заметить, что накладные расходы или время, которое требуется для автоматического анализа дампов, пропорционально объему оперативной памяти, так как размерами кэш-памятей можно пренебречь.

Работа с дампом памяти осуществляется посредством пакета программ, написанных на языке программирования PERL. В пакет входят программа для поиска и сличения когерентных данных и программа для вывода результата. Сравнение данных происходит автоматически после сброса дампа, и сразу же выводится результат о найденных не сличениях.

Предложенная методика функционирует с произвольными тестовыми векторами, однако, для повышения качества тестирования и сокращения времени, требуемого для расширения существующей базы тестовых векторов, был разработан генератор направленных псевдослучайных тестов с управляемыми характеристиками. Генератор создает файлы объектного кода, трассы команд и операндов и имеет ряд средств управления вероятностями появления событий в созданной программе. Так как полученный в результате генерации код должен максимально загружать и использовать все возможности подсистемы памяти 2-х процессорной конфигурации, он должен состоять из последовательностей запросов по считыванию/прописи в память, причём требуется использовать пересекающиеся адреса для моделирования работы аппаратуры с общими данными. Также существенной особенностью генератора явилось создание интенсивного потока заявок к памяти с вытеснением строк в кэш-памятях для максимального использования архитектурной производительности микропроцессора и, тем самым, улучшая полноту проверок. Следует заметить, что первоначально данный генератор возник не в роли полигона для испытания данного метода, а как самостоятельное средство, он не несет в себе никаких проверок, но позволяет обнаружить заторы (DEADLOCK) или неучтенные состояния в аппаратуре. Все это делает генератор необходимым средством для использования метода проверок протоколов когерентности.

Анализ результатов прохождения тестовой программы требует сохранить состояние памяти, для этого средствами языка VERILOG, на котором описан объект верификации, был подключен дополнительный модуль для копирования (дампа) данных из памяти моделируемого объекта на файловую систему. Дамп данных осуществлялся по команде успешного завершения тестовой программы, следует также заметить, что сброс дампа не запускается, пока не закончились все операции с памятью, иначе возможны ситуации, нарушающие когерентность.

Данный метод является достаточно эффективным средством для поиска ошибок в работе аппаратуры, отвечающей за протокол поддержки когерентности, но имеет ряд недостатков,

один из них это сложность в отладке найденных ошибок. Для упрощения процесса отладки была написана программа, позволяющая локализовывать найденные ошибки во времени и, как следствие, команду, вызвавшую запрос в кэш-память.

2.3. Результаты

Результаты использования данного метода зависят от тестовых векторов, для которых он может применяться. Особенное увеличение найденных ошибок мы можем наблюдать на генераторе псевдослучайных тестов. Например, если использовать генератор тестов, направленных на работу с подсистемой памяти, то при подключении сличения когерентности кэш-памятей увеличивается количество обнаруженных ошибок, связанных с нарушением целостности состояния памяти по сравнению с традиционным тестированием. В таблице показаны практические результаты прохода пакета тестов (3517 тестовых векторов) с включенным/выключенным режимом проверки протоколов когерентности. Генератор был отконфигурирован так, чтобы полученные тестовые вектора имели одинаковую длину. Из таблицы видно, что при использовании данного метода, как средства дополнительного мониторинга памяти, количество выявленных тестов с неправильно работающим тестовым вектором резко возрастает, причем прирост существенный, около 10%, что говорит о достаточной эффективности данной методики.

Параметр	С использованием данного метода	Без использования данного метода
Кол-во тестов	3517	
Кол-во не прошедших тестов	64	57
Время прохода всех тестов	59 мин 14 сек	58 мин 47 сек

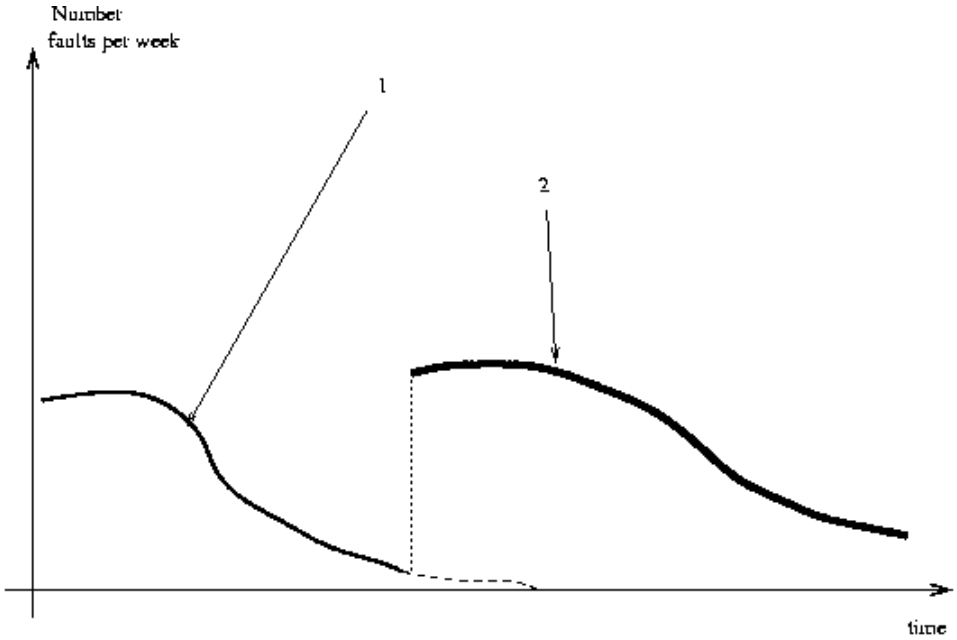

Рис. 2. Эффективность проверки протокола когерентности

Также прилагается график, показывающий данную эффективность (Рис. 2), на графике показана зависимость скорости поиска ошибок от времени (скорость поиска ошибок выражена в количестве ошибок, обнаруженных за неделю), время - в неделях. Первая кривая (более тонкая) - относится к использованию генератора без проверки когерентности, более жирная кривая - это дальнейшее использование того же генератора, но уже с использованием проверки дампов памяти. Из графика следует, что на начальном этапе (линия 1) количество обнаруженных ошибок возрастает, вследствие начала тестирования (влияют переходные процессы, такие как отладка самого генератора). Далее кривая переходит в прямую, наступает момент, когда ситуа-

ции, предусмотренные тестовыми программами, исчерпываются. Включение режима проверки протоколов когерентности (линия 2) приводит к тому, что на том же наборе тестов обнаруживаются дополнительные ошибки, которые были пропущены при прямом тестировании.

Заключение

В работе представлен подход к верификации подсистемы памяти, основанный на проверке когерентности кэш-памятей всех уровней. Эксплуатация этого метода показала большую эффективность и малый процент накладных расходов и успешно применяется при верификации микропроцессора.

Авторы выражают благодарность А. Супруну за предоставленные средства трассировки запросов к кэш-памятям процессора.

Литература

1. С.С. Гилязов, В.В. Рудомётов, Ю.Х. Сахин. Функциональная верификация конвейерного микропроцессора. Научная конференция, посвященная 70-летию со дня рождения академика В.А. Мельникова, сборник докладов, Москва – 1999 г.
2. Рудомётов В.В., Ефремова О.А., Семенов В.С., Использование анализа содержимого регистрового файла для верификации конвейера в микропроцессоре с широким командным словом, XXVII Гагаринские чтения, молодежная научная конференция, тезисы докладов, Москва – 2001 г.
3. Daniel J. Sorin, Manoj Plakal, Anne E. Condon, Mark D. Hill, Milo M.K. Martin, David A. Wood, Specifying and verifying a broadcast and multicast snooping cache coherence protocol, IEEE transactions on parallel and distributed systems, Vol. 13, No. 6, June 2002