УДК 004.052.42 DOI: 10.31114/2078-7707-2020-3-57-64

Функциональная верификация атомарных операций с использованием анализа конфликтов блокировок

Н.А. Гревцев, П.А. Чибисов

ФГУ ФНЦ НИИСИ РАН, г. Москва

ngrevcev@cs.niisi.ras.ru, chibisov@niisi.ras.ru

***Аннотация* — Известно, что симметричный мультипроцессор (SMP) должен использовать набор механизмов синхронизации для достижения результатов, свободных от состояния гонки, а также для корректного распределения ресурсов. Было доказано, что использование атомарных операций для доступа к разделяемым областям памяти в SMP-системах является основным методом предотвращения потери данных при реализации в программном обеспечении таких примитивов, как mutual exclusion, spinlock, thread execution barrier. Современные микропроцессорные архитектуры предоставляют различные виды атомарных операций, например, инструкции LL и SC на микропроцессорах с архитектурой MIPS используются в связке, чтобы гарантировать атомарность доступа к общей памяти для операций чтения и записи.**

Механизмы синхронизации, такие как критические секции и барьеры, используемые для обеспечения исключительного доступа к разделяемым ресурсам, являются потенциальным узким местом производительности в многопоточных приложениях. Существует множество подходов, предлагающих различные методы для анализа и уменьшения влияния этой проблемы в программном или аппаратном обеспечении. Описанные подходы количественно измеряют накладные расходы на механизмы синхронизации, а также оценивают влияние данных примитивов синхронизации на общее время выполнения многопоточных приложений.

Все упомянутые исследования и описанный в литературе опыт могут быть использованы для расширения покрытия функциональной верификации атомарных операций. Функциональная верификация атомарных операций является достаточно трудоемким процессом, поскольку корректность работы атомарных операций не может быть проверена с помощью стохастических методов генерации тестов в силу непредсказуемости их поведения. Об этом свидетельствует отсутствие достаточной информации по данной теме, что с одной стороны могло бы означать, что атомарные операции были успешно проверены традиционными методами. Однако можно найти по крайней мере две ссылки на задокументированные ошибки известных процессоров, в которых перечислены проблемы, связанные с неправильным поведением атомарных операций.

Представленный в статье анализ механизмов синхронизации является методом, который фокусируется не на вопросах производительности, а на функциональной верификации инструкций атомарных операций, которые обеспечивают основу механизмов синхронизации.

***Ключевые слова* — lock contention, cache contention, атомарные инструкции, функциональная верификация, псевдослучайная направленная генерация тестов, тесты производительности, PARSEC, lock torture, тестовое покрытие.**

I. Введение

Функциональная верификация атомарных операций является довольно длительным и трудоемким процессом, поскольку атомарные операции не могут быть проверены с помощью стохастических методов генерации тестов из-за непредсказуемого результата выполнения атомарных операций. В следствии чего, эталонный эмулятор, используемый в традиционном маршруте верификации для сравнения результатов выполнения тестов, не применим при тестировании данной области микроархитектуры. Для создания новой методики тестирования атомарных инструкций были использованы методы анализа и прогнозирования конфликтов блокировки. Представленная статья является продолжением исследования, посвященного верификации атомарных операций [28].

Обычно принято избегать совместного использования данных при написании многопоточных программ [1]. Например, ложное разделение данных - это широко известная проблема в многопоточных приложениях, которая может значительно уменьшить как производительность, так и масштабируемость [2], [3], [4]. Тем не менее, различные виды синхронизации между потоками, такие как барьеры или критические секции необходимы, следовательно, нельзя избежать конфликтов при получении исключительного доступа к разделяемым ресурсам (это называется истинным совместным использованием). Использование атомарных операций для доступа к областям общей памяти в SMP-системах является основным методом предотвращения повреждения данных. Атомарные операции – это примитивы синхронизации самого низкого уровня. Они используются в качестве строительных блоков для конструкций более высокого уровня, таких как, например, барьеры, *mutual exclusion*, *spinlock* (рис. 1).

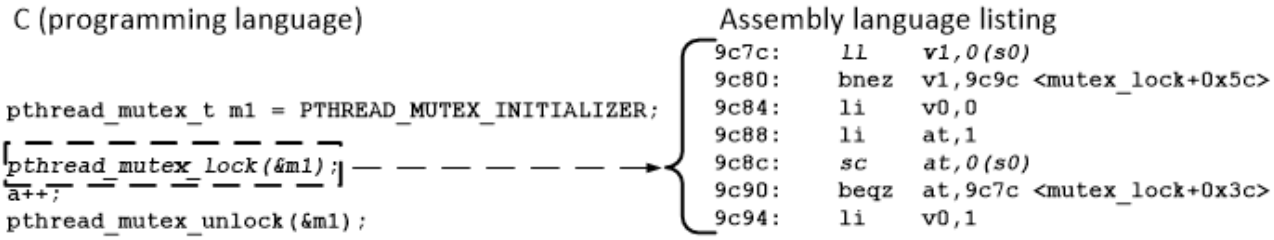

Рис. 1. Пример атомарных операций в программном коде

Известно, что механизмы синхронизации, такие как критические секции, также подвержены влиянию конфликтов когерентности данных [5].

Существует множество подходов, предлагающих различные методы для анализа и уменьшения этих проблем в программном или аппаратном обеспечении. Эти подходы количественно определяют накладные расходы на выполнение механизмов синхронизации или оценивают влияние, которое эти примитивы оказывают на время выполнения многопоточных приложений [6–15]. Тем не менее, разумно создавать тестовые ситуации с совместным использованием областей данных всеми потоками, если требуется проверить взаимодействие между ядрами. Ключевым аспектом представленной статьи является то, что все предыдущие исследования и опыт могут быть использованы для увеличения охвата функциональной верификации атомарных операций.

II. АТОМАРНЫЕ ОПЕРАЦИИ НА ПРОГРАММНО-АППАРАТНОМ УРОВНЕ

Согласно документации MIPS Architecture [16], связка инструкций LL (*Load Linked*) и SC (*Store Conditional*) предоставляет примитив для реализации атомарных операций чтение-модификация-запись (RMW) для областей памяти. Когда инструкция LL выполняется, она запускает активную RMW последовательность, заменяя любую другую последовательность, которая была активной. Последовательность RMW завершается последующей инструкцией SC, которая либо завершает последовательность RMW атомарно и успешно, либо завершается неудачно. Разрыв последовательности при выполнении LL и SC происходит при любом из следующих событий: когерентная запись другим ядром процессора в совпадающий блок физической памяти или выполнение команды «eret» (exception return). LL и SC используются для атомарного обновления областей памяти, как показано ниже:

L1: LL T1, (T0)	# Load Linked — считать счетчик
ADDI T2, T1, 1	# операция инкремента
SC T2, (T0)	#Store Conditional – попытка сохранения, с проверкой на атомарность
BEQ T2, 0, L1	# если операция не атомарна (0), повтор попытки
NOP	# слот задержки

Приведенный программный код является базовым примером, который используется для конструирования таких примитивов, как барьеры, mutual exclusion, spinlock.

Процесс тестирования начинается с создания простых рукописных тестов на языке ассемблера, направленных на проверку основных аспектов выполнения атомарных инструкций. Даже простой пример рукописного теста должен учитывать все соответствующие факторы, которые могут привести к ошибкам и несоответствиям. Конечно, невозможно провести исчерпывающий поиск по всем возможным комбинациям тестов вручную. Поэтому первый вопрос - как провести эвристический поиск и создать автоматизированную или частично автоматизированную систему генерации тестов. Второй вопрос еще более значим: атомарные операции не могут быть проверены с помощью стохастических методов генерации тестов из-за их непредсказуемой природы. Это означает, что эталонный эмулятор (ISS), который обычно используется для получения ожидаемых результатов выполняемого теста, неприменим. Поэтому было предложено использовать один из подходов самопроверки.

На ранних этапах цикла разработки RTL-модели были использованы простые рукописные тесты на языке ассемблера с внутренней самопроверкой. Простейшими примерами параллельных программных алгоритмов с гонками данных, которые могут быть реализованы без поддержки библиотек операционной системы, являются операции с массивами и связанные списки [1]. Эти алгоритмы были выбраны по следующим причинам: 1) они представляют собой фрагменты реальных вычислительных задач; 2) их можно вручную разделить на произвольное число параллельных потоков; 3) можно легко назначить критические секции кода для защиты переменных данных, которые связаны и не могут делиться между потоками; 4) адреса блокировок (рассматриваемых атомарных инструкций) могут быть намеренно смешаны с переменными структур данных, чтобы увеличить число конфликтов блокировок. Кроме того, обнаружено, что генерация тестов таких тестов должна быть частично или полностью автоматизирована для успешной функциональной проверки пары команд LL и SC. Ситуации истинного и ложного разделения данных могут также быть упомянуты среди соответствующих факторов, которые оказывают сильное влияние на выполнение инструкций LL и SC.

Для увеличения покрытия тестирования атомарных операций в течении функциональной верификации RTL-модели требуется расширить имеющиеся подходы многоядерного направленного стохастического тестирования. Решение в значительной степени основывается на результатах предыдущих исследований [7], [9], посвященных вопросам конкуренции за распределенные ресурсы (например, кэш-память и память), конкуренции за программные ресурсы (для блокировок и барьеров потоков), издержек распараллеливания и дисбаланса рабочей нагрузки [17]. Главным образом, требуется сосредоточиться на понимании конфликтов блокировок. В параллельной программе использование общих данных обычно защищено блокировками, чтобы гарантировать исключительный доступ. Если несколько потоков пытаются получить доступ к совпадающим данным, только один поток может получить доступ в один момент времени, остальные вынуждены ждать вместо выполнения полезной работы [7]. Это называют конфликтом блокировки [7], [8].

Механизмы синхронизации, такие как критические секции, используемые для обеспечения исключительного доступа к критическим ресурсам и структурам данных, являются хорошо известными потенциальными узкими местами производительности в многопоточных приложениях [10]. Было изучено множество подходов, предлагающих различные методы, позволяющие обнаружить и сократить влияние примитивов синхронизации в программном или аппаратном обеспечении. Эти подходы позволяют количественно оценить накладные расходы на выполнение механизмов синхронизации или оценить влияние, которое эти примитивы оказывают на время выполнения многопоточных приложений. В отличие от этих подходов, вместо стремления к увеличению производительности в статье рассматривается концепция состязания за блокировку (аналогичные термины «конкуренция за потоки» или «конкуренция за кэш-память») в контексте создания нового метода повышения качества функциональной верификации атомарных операций.

Следует подчеркнуть, что концепция конкуренции за блокировку напрямую связана с атомарными инструкциями (рис. 2). Однако, с точки зрения программиста, конфликт блокировок – это условие, которое: а) ограничивает степень параллелизма путем сериализации доступа к защищенным общим данным или б) связано с бездействием в выполнении потока. Обычно эти условия упоминаются как причины снижения производительности, и считается важным минимизировать суммарное количество атомарных операций, которые требуются во время критических секций [9].

Несмотря на то, что при написании параллельных программ и драйверов ядра ОС рекомендуется избегать конфликтов данных, предлагаемый в статье подход, напротив, заключается в создании случайных тестовых ситуаций с разделяемыми областями данных между потоками, для намеренного повышения интенсивности взаимодействия между атомарными инструкциями в более короткий период времени.

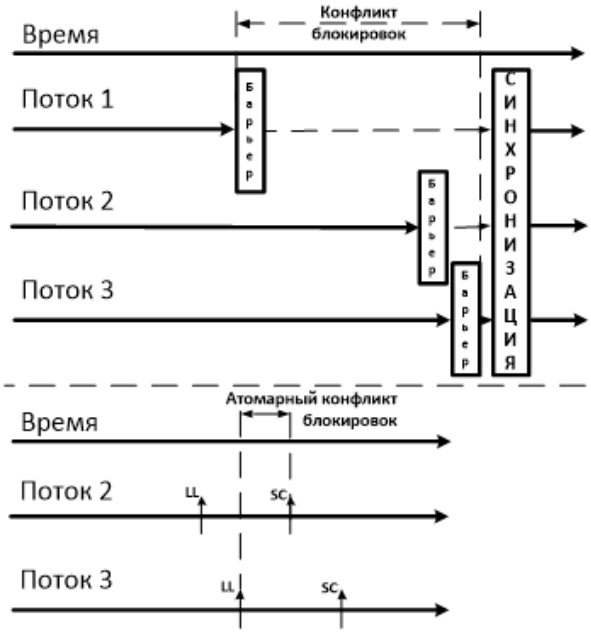

Рис. 2. Конфликт блокировки и атомарный конфликт блокировки на примере барьера

Предлагаемое определение конфликтов отличается от общепринятого. Необходимо создать конфликты атомарных операций таким образом, чтобы полностью проверить их функциональность. Определим этот тип конкуренции (для простоты возьмем только два ядра) как совпадение двух одновременно выполняемых пар инструкций Load Linked и Store Conditional на разных ядрах процессора, если они используют один и тот же физический адрес и одна из инструкций Store Conditional прерывает выполнение.

Результаты исследований показывают, что вероятность конфликта LL-SC на несколько порядков меньше, чем вероятность возникновения общего конфликта кэш-памяти или блокировки. Это может быть объяснено тем фактом, что конфликт LL-SC возникает как следствие конфликта кэш-памяти (рис. 3), но вероятность разрыва связки LL-SC значительно меньше. Также стоит отметить, что число уникальных обращений к памяти, сделанных между двумя атомарными обращениями к одной и той же ячейке памяти, представляет особый интерес для исследования [18]. Этот метод, называемый анализом расстояний повторного использования (reuse distance), может быть эффективен для измерения локальности данных атомарных операций [19].

III. Экспериментальная оценка существующих нагрузочных тестов с точки зрения возникновения конфликтов блокировок

Этот раздел отвечает на следующие вопросы:

1) Какова связь между конфликтом блокировок и конфликтом атомарных операций с точки зрения времени выполнения?

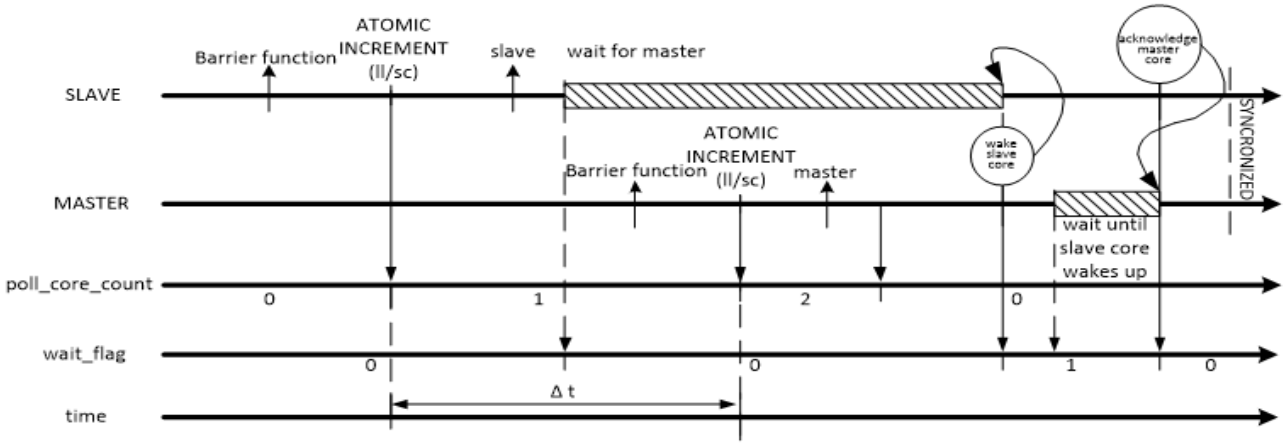

Рис. 3. Общий вид барьерной функции

2) Какова вероятность выполнения атомарных операций при выполнении группы репрезентативных тестов?

3) Как оценить частоту возникновения разрыва связки Load Linked - Store Conditional?

4) Как использовать полученную информацию для улучшения качества функциональной верификации инструкций Load Linked и Store Conditional?

Чтобы ответить на эти вопросы, используются реальные приложения из современного набора многопоточных программных тестов PARSEC 2.1 [20]. Репрезентативная часть каждого теста выполняется в MIPS Instruction Set Simulator, чтобы получить трассировку инструкций достаточной длинны. Было принято во внимание, что для каждого бенчмарка существует так называемая область интереса (region of interest, ROI) [21], которая указывает, какая часть бенчмарка выполняется параллельно. Параметр ROI также важен для гарантии того, что результаты, полученные при моделировании, отражали реальное поведение программы.

Выполняя моделирование только области ROI с набором входов simmedium (который подходит для микроархитектурных исследований на симуляторах [22]), удалось существенно сократить время моделирования.

Представленная далее стратегия измерения частоты разрывов LL-SC была оценена на пакете тестов Linux Kernel Lock Torture Test [23], который можно запустить как часть Linux Test Project (LTP) [24]. Этот стресс-тест состоит из нескольких потоков ядра, которые получают блокировку и удерживают ее в течение определенного времени, таким образом моделируя различное поведение критической секции. Степень конкуренции за блокировку может быть смоделирована путем увеличения времени удержания этой критической области и / или путем создания большего количества kthreads [23].

Во-первых, были последовательно изучены ключевые характеристики тестов PARSEC, особенно разнообразие примитивов синхронизации. Во-вторых, выбраны 5 эталонных тестов и некоторые другие пользовательские приложения, известные как наиболее высоко нагруженные с точки зрения количества блокировок [20]. В-третьих, было проведено моделирование и получены результаты тестов с точки зрения поведения инструкций LL и SC (Таблица 1), а также рассмотрена вариативность структур примитивов синхронизации. Полученные знания о конфликтах блокировок могут помочь понять, как расширить методы генерации случайных тестов для проверки атомарных инструкций.

Эксперименты были проведены на 2-ядерном симуляторе архитектуры MIPS под управлением Debian GNU / Linux 9 «stretch». Был применен gcc-6.3.0 с оптимизацией «-O3» с ядром Linux 4.15. Во всех экспериментах с тестами PARSEC используются входные наборы simmedium. В ходе эксперимента моделируются только ROI каждого теста и используется трассировка инструкций длиной 10^8 инструкций процессора для каждого ядра, которая содержит всю необходимую информацию. Было получено соотношение команд загрузки и сохранения, количество атомарных инструкций (LL и SC), уникальные атомарные инструкции, выполняемые только на одном ядре (их программные счетчики, PC), общие и частные адреса для атомарных команд, а также были рассмотрены события, приводящие к разрыву связки LL-SC. Результаты оценки поведения команд LL и SC показаны в табл. 1.

Важно изучить все типичные блоки программного кода, содержащие атомарные инструкции, и типы операций, которые вызвали сбой SC. Однако вероятность сбоя инструкций SC, наблюдаемых в сложных реальных приложениях, запускаемых на симуляторе, показывает, что условия гонки и сбои SC крайне редки для такой корреляции общих адресов.

Таблица 1

Оценка результатов тестов с точки зрения поведения инструкций LL и SC

Название теста		Total Reads (M)	Total Writes (M)	Total LL executed	Total SC executed	Unique PCs of LLs	LL unique access addresses	Private/ Shared LL addresses	SC Failed (LL-SC Broken)
PARSEC	streamcluster	55.08	0.65	6,695	6,033	18	17	13/4	122
	fluidanimate	54.83	10.46	219,779	219,770	37	3449	1,914/1,535	4
	bodytrack	38.49	12.28	129,893	129,866	123	187	141/46	0
	blackscholes	3.51	1.46	91,452	91,450	53	82	82/0	0
	ferret	47.80	15.96	23,677	23,515	23,666	637	549/66	5
LTP: locktorture	spin_lock	40.17	17.20	1,563,381	1,559,146	54	82	44/38	591
	spin_lock_irq	39.12	17.05	1,493,647	1,489,073	19	42	23/19	483
	rw_lock	17.99	3.10	478,720	476,367	30	40	23/17	2
	rw_lock_irq	15.31	4.76	865,469	864,679	28	40	32/8	0
	mutex_lock	8.78	2.67	530,696	530,688	30	54	42/12	0
	rwsem_lock	7.60	2.31	459,437	459,437	18	36	32/4	0
	rtmutex_lock	34.29	17.05	2,028,869	1,886,839	13	21	18/3	0
	ww_mutex_lock	14.75	4.47	892,719	892,717	24	35	34/1	0
	percpu_rwsemlock	12.12	3.68	732,757	732,756	23	45	26/19	0

Предлагаемая стратегия, напротив, предназначена для преднамеренного увеличения конфликтов LL-SC при создании и запуске случайных проверочных тестов для RTL-модели многоядерного процессора.

IV. Метод повышения содержания атомарных операций в случайных тестах

После анализа поведения тестов авторы пришли к выводу, что для эффективной проверки этой области проектирования микропроцессоров, необходимо решить три основные задачи:

- значительное уменьшение расстояния повторного использования (временная локализация) [18];
- увеличение процента общих адресов (пространственная локализация);
- диверсификация циклов LL-SC.

Все эти задачи могут быть решены с помощью направленной генерации случайных тестов. Важно понимать, что ошибки в поведении RTL-модели обычно связаны с сочетанием многих факторов, таких как попадания или промахи в кэш-памяти всех уровней, типы зависимости регистров, поведение блока предсказания ветвлений и т. д. Поэтому один из показателей тестового покрытия связан с вариативностью циклов LL-SC. Тем не менее, следует отметить, что полностью случайно (произвольно) сгенерированный цикл LL-SC может быть либо по своей природе невыполнимым (может вызвать динамическую блокировку), либо результат его выполнения будет непредсказуем.

Полученные уникальные комбинации циклов LL-SC (см. табл. 1) могут быть использованы для дальнейшего процесса тестирования. Предполагается, что набор тестов, представленных выше, охватывает большинство способов использования атомарных операций. Все полученные комбинации могут быть классифицированы в соответствии с их структурой (с точки зрения типов инструкций и их зависимостей).

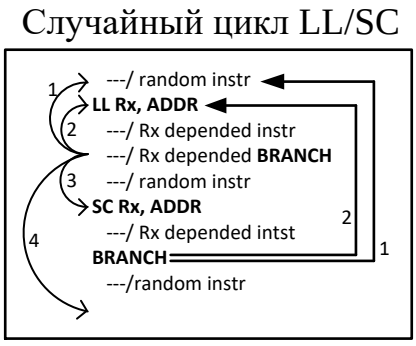

Рис. 4. Формализованный LL-CS цикл

Чтобы охватить все оставшиеся возможные ситуации, результирующий набор тестовых ситуаций может быть формализован и представлен в виде псевдокода, показанного на рис. 4.

Цикл всегда состоит из инструкций LL и SC, обращающихся к одному и тому же адресу, и инструкции перехода в зависимости от результата выполнения инструкции SC. Пространство между LL и SC может быть

заполнено произвольным количеством случайных инструкций с некоторыми ограничениями.

Чтобы направить генератор к интересным событиям (определенным областям в дизайне или некоторому конкретному сценарию выполнения атомарных инструкций), в генератор вносится интеллект, то есть экспертное знание инженера-верификатора (*testing knowledge*) [25]. Создаваемые генератором ситуации могут быть бессмысленны с точки зрения программиста, потому что они не могут возникнуть в реальных приложениях. В то же время они позволяют найти ошибки архитектуры, увеличивая сложность и разнообразие тестовых ситуаций. Комбинации всех циклов LL-SC, генерируемых по предлагаемым моделям, обеспечивают 100% покрытия сценариев использования LL-SC.

Чтобы обеспечить правильное функционирование атомарных операций, необходимо проверить, является ли сбой SC корректным в зависимости от различных типов когерентных записей в любой момент времени. Поскольку эту проблему невозможно решить с помощью комбинаторных тестов, направленный генератор псевдослучайных тестов с достаточно большой рандомизацией может служить инструментом для создания случайных сбоев атомарности LL-SC. Текущий тестовый генератор Ristretto [26] был расширен для создания тестовых случаев со случайными комбинациями загрузки / сохранения и атомарных инструкций для генерации необходимых входных воздействий. Кроме того, потоки, созданные генератором, могут одновременно использовать совместные ресурсы памяти для инициирования взаимодействий и когерентных транзакций на системной шине. Также есть возможность направлять процесс тестирования, изменяя вероятность ложного совместного использования данных, минимальные и максимальные размеры областей памяти и частоту появления макросов в тесте.

Псевдослучайные пары LL-SC, полученные на предыдущем шаге, добавляются в шаблон теста в виде макросов и имеют такое же распределение памяти, что и обычные инструкции загрузки / сохранения. Поскольку каждый поток имеет доступ к своим собственным байтам в строке кэш-памяти (и в области памяти) во время выполнения подтеста, конечное состояние сегмента памяти после выполнения цикла LL-SC является детерминированным [27]. В то же время обращения к памяти из разных потоков могут принадлежать совпадающим строкам кэш-памяти, а тестовые последовательности обращений к памяти имеют случайный характер. Это приводит к тому, что вероятность сбоя SC зависит от временной локализации и пространственной локализации, которые задаются в шаблоне случайного теста. Следовательно, общая частота конфликтов LL-SC зависит от качества генератора псевдослучайных тестов и от распределения памяти между потоками.

Пример теста, полученного представленным способом, а также распределение памяти между потоками показан на рис. 5. Очевидно, что каждый атомарный макрос в таком случае имеет очень маленькое расстояние повторного использования (т.е. высокую частоту появления в трассе теста) из-за большого количества запросов от другого ядра к совпадающей строке кэш-памяти. Гарантируется, что любой тип цикла LL-SC из наиболее высоконагруженных с точки зрения конфликтов кэш-памяти приложений пользователя охватывается сгенерированными тестовыми примерами в разных контекстах выполнения.

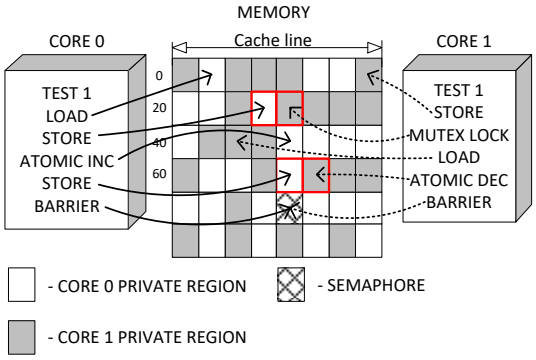

Рис. 5. Распределение памяти между потоками

Все критические ошибки, найденные этим методом, можно разделить по нескольким группам:

- инструкция SC сообщила об успехе, но не записала значение в память;
- инструкция SC сообщила об отмене исполнения, но записала значение в память;
- инструкция SC не распознала когерентную запись от другого ядра и записала значение в память.

V. Заключение

В статье изучаются причины и вероятность возникновения конфликтов блокировок в многопоточных приложениях пользователя. Результаты исследования временной локальности и пространственной локальности атомарных операций позволяют увеличить вероятность конфликта блокировок в создаваемых тестовых сценариях. Представленная стратегия, в отличии от многих предыдущих исследований, основана на намеренном увеличении числа конфликтов атомарных пар инструкций LL-SC при создании случайных тестов для RTL-модели многоядерного процессора. В статье доказывается, что предложенная новая методика будет охватывать все возможные сценарии использования атомарных инструкций, которые можно найти в реальных приложениях.

Предложенный подход был успешно применен для верификации RTL-модели двухъядерного микропроцессора с архитектурой SMP, разработанной в НИИСИ РАН. Этот метод позволил обнаружить большинство ошибок согласованности памяти, остановок конвейера, взаимоблокировок и других случаев ошибочного поведения атомарных операций. Преимущество метода заключается в отсутствии необходимости изменять стратегию и процесс верификации для адаптации к новому проекту. Кроме того, возможно нахождение ошибки автоматически из-за стохастической природы генератора.

Литература

[1] M. Herlihy and N. Shavit. The Art of Multiprocessor Programming, Revised Reprint. Elsevier, 2012.

[2] T. Liu et al., "PREDATOR: Predictive false sharing detection", PPoPP 2014.

[3] Tongping Liu , Xu Liu, Cheetah: detecting false sharing efficiently and effectively, Proceedings of the 2016 International Symposium on Code Generation and Optimization, March 12-18, 2016.

[4] Zhao, Q., Koh, D., Raza, S., Bruening, D., Wong,W.-F., Amarasinghe, S. Dynamic Cache Contention Detection in Multi-threaded Applications. In Proceedings of the international conference on Virtual Execution Environments, VEE'11, pages 27–38, 2011.

[5] A. Starke. Locking in os kernels for smp systems. Citeseer, 2006.

[6] Xu, C., Chen, X., Dick, R.P., Mao, Z.M.: Cache Contention and Application Performance Prediction for Multi-Core Systems. In: Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Softwares IEEE ISPASS 2010, White Plains NY, USA, pp. 76–86.

[7] X Pan, J Lindén, B Jonsson Predicting the Cost of Lock Contention in Parallel Applications on Multicores using Analytic Modeling - MCC12, 2012.

[8] R. Gu, G. Gin, L. Song, L. Zhu, and S. Lu, "What change history tells us about thread synchronization", in FSE, August 2015.

[9] N. R. Tallent, J. M. Mellor-Crummey, and A. Porterfield, "Analyzing lock contention in multithreaded applications," in PPoPP, February 2010.

[10] G. Chen, P. Stenstrom Critical lock analysis: Diagnosing critical section bottlenecks in multithreaded applications. In Proceedings of Supercomputing: the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), pages 71:171:11, Nov. 2012.

[11] H. Ding, X. Liao ; H. Jin ; X. Lv ; R. Guo Reducing lock contention on multi-core platforms. In Proceedings of 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014.

[12] Gramoli, V. More than you ever wanted to know about synchronization: synchrobench, measuring the impact of the synchronization on concurrent algorithms. In Proceedings of 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, pp. 1–10.

[13] Kempf S., Veldema R., Philippsen M. (2013) Compiler-Guided Identification of Critical Sections in Parallel Code. In: Jhala R., De Bosschere K. (eds) Compiler Construction. CC 2013. Lecture Notes in Computer Science, vol 7791. Springer, Berlin, Heidelberg.

[14] S. Dutta, S. Manakkadu, and D. Kagaris, "Classifying performance bottlenecks in multi-threaded applications," in MCSoc 2014, September 2014.

[15] Sahelices B., Ibáñez P., Viñals V., Llabería J.M. (2009) A Methodology to Characterize Critical Section Bottlenecks in DSM Multiprocessors. In: Sips H., Epema D., Lin HX. (eds) Euro-Par 2009 Parallel Processing. Euro-Par 2009. Lecture Notes in Computer Science, vol 5704. Springer, Berlin, Heidelberg.

[16] MIPS64™Architecture For Programmers, V.2:The MIPS64™ Instruction Set Reference Manual, Revision 6.04, MIPS Technologies Inc., 2015, 551 pp.

[17] Jungju Oh, Christopher J. Hughes, Guru Venkataramani, and MilosPrvulovic. 2011. LIME: A Framework for Debugging Load Imbalancein Multi-threaded Execution. InProceedings of the 33rd InternationalConference on Software Engineering (ICSE). 201–210.

[18] K. Beyls, E. D'Hollander Reuse Distance as a Metric for Cache Behavior. In Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, IASTED, 2001. p.617-622.

[19] R. Hemani, S. Banerjee, A. Guha, "Accord: An analytical cache contention model using reuse distances for modern multiprocessors", 21st Annual International Conference on High Performance Computing Student Research Symposium (HiPC SRS), December 2014.

[20] Bienia, S. Kumar, J. P. Singh, K. Li, The parsec benchmark suite: Characterization and architectural implications, in: Proceedings of the17th International Conference on Parallel Architectures and Compilation Techniques, 2008.

[21] G. Southern and J. Renau. Analysis of PARSEC workload scalability. In 2016 IEEE International Symposium on Performance Analysis ofSystems and Software (ISPASS), pages 133–142, April 2016.

[22] A Memo on Exploration of SPLASH-2 Input Sets PARSEC Group Princeton University, 2011.

[23] Kernel Lock Torture Test Operation[Electronic resource] https://www.kernel.org/doc/Documentation/locking/locktorture.txt

[24] Modak, Singh, YAMATO Putting LTP to Test - validating the Linux Kernel and test cases. Proceedings of the 2009 Montreal Linux Symposium, July 2009, Montreal, Canada.

[25] IBM: Katz, Yoav, Rimon, Michal, Ziv, Avi and Shaked, Gai. "Learning microarchitectural behaviors to improve stimuli generation quality." Paper presented at the meeting of the DAC, 2011.

[26] Ristretto: random test generator for multicore microprocessor cache coherence verification A.V. Smirnov, P.A. Chibisov SELECTED ARTICLES of the VIII All-Russia Science&Technology Conference MES-2018 Part 1

[27] Multicore Processor Models Verification in the Early Stages N.A. Grevtsev, P.A. Chibisov SELECTED ARTICLES of the VIII All-Russia Science&Technology Conference MES-2018 Part1

[28] N.A. Grevtsev, P.A. Chibisov, "A novel technique for atomic instructions functional verification using lock contention analysis," 2019 IEEE East-West Design & Test Symposium (EWDTS).

Atomic Instructions Random Tests Generation Using Lock Contention Analysis

N.A. Grevtsev, P.A. Chibisov

SRISA RAS, Moscow

ngrevcev@cs.niisi.ras.ru, chibisov@niisi.ras.ru

***Abstract* — Atomic operations functional verification is known to be a rather labour-intensive process because atomic operations cannot be tested with the help of stochastic test generation methods due to their unpredictable nature.**

Atomic operations are synchronization primitives used as building blocks for mutual exclusions, spinlocks, and thread execution barriers.
There are numerous approaches proposing a variety of methods to analyze and reduce lock contention problems in software or in hardware. These approaches quantify the synchronization mechanisms execution overhead or assess the impact these primitives have on the completion time of multithreaded application.
However, it is reasonable to create test cases with forced shared data areas between threads intentionally to test the interaction between cores. In the paper, we propose to apply all previous researches to increase the coverage of atomic operations functional verification.
According to the mentioned methods, we run PARSEC benchmarks and Linux Kernel Lock Torture Test to estimate and find the most contented segments of code in these applications. The obtained knowledge has been applied for extension of our current test generator Ristretto to create test cases with random load/store and atomic instructions combinations to generate necessary stimulus.
The discussed approach was successfully tried out in practice on the verification process of the RTL-model of dual-core microprocessor with SMP developed in SRISA RAS. The advantage of the technique is that there is no need to change verification strategy and process to adjust to a new project design.
***Keywords* — lock contention, cache contention, atomic instructions, functional verification, random test generation, PARSEC benchmark, lock torture, test coverage, simulator.**

REFERENCES

[1] M. Herlihy and N. Shavit. The Art of Multiprocessor Programming, Revised Reprint. Elsevier, 2012.

[2] T. Liu et al., "PREDATOR: Predictive false sharing detection", PPoPP 2014.

[3] Tongping Liu , Xu Liu, Cheetah: detecting false sharing efficiently and effectively, Proceedings of the 2016 International Symposium on Code Generation and Optimization, March 12-18, 2016.

[4] Zhao, Q., Koh, D., Raza, S., Bruening, D., Wong,W.-F., Amarasinghe, S. Dynamic Cache Contention Detection in Multi-threaded Applications. In Proceedings of the international conference on Virtual Execution Environments, VEE'11, pages 27–38, 2011.

[5] A. Starke. Locking in os kernels for smp systems. Citeseer, 2006.

[6] Xu, C., Chen, X., Dick, R.P., Mao, Z.M.: Cache Contention and Application Performance Prediction for Multi-Core Systems. In: Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Softwares IEEE ISPASS 2010, White Plains NY, USA, pp. 76–86.

[7] X Pan, J Lindén, B Jonsson Predicting the Cost of Lock Contention in Parallel Applications on Multicores using Analytic Modeling - MCC12, 2012.

[8] R. Gu, G. Gin, L. Song, L. Zhu, and S. Lu, "What change history tells us about thread synchronization", in FSE, August 2015.

[9] N. R. Tallent, J. M. Mellor-Crummey, and A. Porterfield, "Analyzing lock contention in multithreaded applications," in PPoPP, February 2010.

[10] G. Chen, P. Stenstrom Critical lock analysis: Diagnosing critical section bottlenecks in multithreaded applications. In Proceedings of Supercomputing: the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), pages 71:171:11, Nov. 2012.

[11] H. Ding, X. Liao ; H. Jin ; X. Lv ; R. Guo Reducing lock contention on multi-core platforms. In Proceedings of 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014.

[12] Gramoli, V. More than you ever wanted to know about synchronization: synchrobench, measuring the impact of the synchronization on concurrent algorithms. In Proceedings of 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, pp. 1–10.

[13] Kempf S., Veldema R., Philippsen M. (2013) Compiler-Guided Identification of Critical Sections in Parallel Code. In: Jhala R., De Bosschere K. (eds) Compiler Construction. CC 2013. Lecture Notes in Computer Science, vol 7791. Springer, Berlin, Heidelberg.

[14] S. Dutta, S. Manakkadu, and D. Kagaris, "Classifying performance bottlenecks in multi-threaded applications," in MCSoc 2014, September 2014.

[15] Sahelices B., Ibáñez P., Viñals V., Llabería J.M. (2009) A Methodology to Characterize Critical Section Bottlenecks in DSM Multiprocessors. In: Sips H., Epema D., Lin HX. (eds) Euro-Par 2009 Parallel Processing. Euro-Par 2009. Lecture Notes in Computer Science, vol 5704. Springer, Berlin, Heidelberg.

[16] MIPS64™Architecture For Programmers, V.2:The MIPS64™ Instruction Set Reference Manual, Revision 6.04, MIPS Technologies Inc., 2015, 551 pp.

[17] Jungju Oh, Christopher J. Hughes, Guru Venkataramani, and MilosPrvulovic. 2011. LIME: A Framework for Debugging Load Imbalancein Multi-threaded Execution. InProceedings of the 33rd InternationalConference on Software Engineering (ICSE). 201–210.

[18] K. Beyls, E. D'Hollander Reuse Distance as a Metric for Cache Behavior. In Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, IASTED, 2001. p.617-622.

[19] R. Hemani, S. Banerjee, A. Guha, "Accord: An analytical cache contention model using reuse distances for modern multiprocessors", 21st Annual International Conference on High Performance Computing Student Research Symposium (HiPC SRS), December 2014.

[20] Bienia, S. Kumar, J. P. Singh, K. Li, The parsec benchmark suite: Characterization and architectural implications, in: Proceedings of the17th International Conference on Parallel Architectures and Compilation Techniques, 2008.

[21] G. Southern and J. Renau. Analysis of PARSEC workload scalability. In 2016 IEEE International Symposium on Performance Analysis ofSystems and Software (ISPASS), pages 133–142, April 2016.

[22] A Memo on Exploration of SPLASH-2 Input Sets PARSEC Group Princeton University, 2011.

[23] Kernel Lock Torture Test Operation[Electronic resource] https://www.kernel.org/doc/Documentation/locking/locktorture.txt

[24] Modak, Singh, YAMATO Putting LTP to Test - validating the Linux Kernel and test cases. Proceedings of the 2009 Montreal Linux Symposium, July 2009, Montreal, Canada..

[25] IBM: Katz, Yoav, Rimon, Michal, Ziv, Avi and Shaked, Gai. "Learning microarchitectural behaviors to improve stimuli generation quality." Paper presented at the meeting of the DAC, 2011.

[26] Ristretto: random test generator for multicore microprocessor cache coherence verification A.V. Smirnov, P.A. Chibisov SELECTED ARTICLES of the VIII All-Russia Science&Technology Conference MES-2018 Part 1

[27] Multicore Processor Models Verification in the Early Stages N.A. Grevtsev, P.A. Chibisov SELECTED ARTICLES of the VIII All-Russia Science&Technology Conference MES-2018 Part 1

[28] N.A. Grevtsev, P.A. Chibisov, "A novel technique for atomic instructions functional verification using lock contention analysis," 2019 IEEE East-West Design & Test Symposium (EWDTS).