УДК 004.052.42 DOI: 10.31114/2078-7707-2021-3-50-56

Генерация синтетических клонов приложений пользователя для функциональной верификации

Н.А. Гревцев, А.А. Краснюк, Д.О. Орлов, П.А. Чибисов

ФГУ ФНЦ НИИСИ РАН, г. Москва, denorl2009@gmail.com

***Аннотация* — В данной работе предлагается методика получения коротких синтетических клонов приложений пользователя, которые могут быть использованы для функциональной верификации RTL-модели микропроцессора. Синтез таких уменьшенных тестов применим при выполнении двух условий: тест воспроизводит заранее заданные характеристики выполнения машинного кода того приложения, на котором он основан, то есть является репрезентативным набором инструкций микропроцессора, и достигает того же результата покрытия за гораздо меньшее время. Множество созданных коротких репрезентативных тестов, время выполнения которых на RTL-модели занимает до 10 минут, используется вместо продолжительных оригинальных программ, запуск которых не представляется возможным за требуемое время.**

***Ключевые слова* — функциональная верификация, метрики покрытия, тесты производительности, граф потока управления, направленная генерация тестов, цикломатическая сложность программ.**

I. Введение

Одной из актуальных задач при разработке микропроцессора (МП) является функциональная верификация его RTL-модели, которая проводится, начиная с самых ранних стадий проектирования. Обычно для тестирования RTL-модели применяются регрессионные базы функциональных тестов, а также стохастические тесты (случайное тестирование, а также генерация на основе ограничений). Как правило, стохастические тесты нацеливают на проверку определенных элементов микроархитектуры МП, однако, полностью случайные тесты не позволяют достичь покрытия «сложно-достижимых» ситуаций. Для достижения нетривиальных ситуаций требуется каким-либо образом учитывать «инженерное (экспертное) знание» при создании шаблонов тестов. К сожалению, такой подход не позволяет гарантировать полноту покрытия.

Опыт тестирования микропроцессоров и их моделей, накопленный за два десятилетия в НИИСИ РАН, позволяет утверждать, что разрабатываемая СБИС должна верифицироваться тестами, отражающими область применения СБИС. Это означает, что для повышения эффективности тестирования из всех потенциальных режимов работы следует в первую очередь выбрать конкретные, нужные пользователям. До передачи RTL-модели в производство следует определить, если это возможно, максимально полный круг задач потенциальных пользователей и направлять тестирование микропроцессора в сторону максимального покрытия в выделенных областях, сужая, если потребуется, область тестирования – бессмысленно создавать тесты режимов, которые не используются ни одним потребителем и не поддержаны в драйверах ОС.

Низкая скорость моделирования RTL-кода является основной причиной невозможности полноценно запустить операционную систему и приложения под ОС, что не позволяет гарантировать отсутствие критических ошибок в уже выпущенном кристалле СБИС микропроцессора. Полноценный запуск операционной системы на рабочих частотах может позволить обнаружить большинство оставшихся в проекте ошибок до выпуска готового изделия. По этой причине даже крупные компании-разработчики МП вынуждены делать дорогостоящий «тестовый» запуск перед полноценным выходом процессора на рынок.

Запуск непродолжительных тестов на RTL-модели не представляет собой сложную задачу, в то время как продолжительные тесты, такие как, например, задачи под ОС, требуют ряда мер по адаптации к запуску. Фактически, требуется решить задачу выборки нескольких репрезентативных частей из продолжительного теста. Цель данной работы – предложить решение вопроса выборки репрезентативных фрагментов из трассы приложения под ОС путём создания синтетических аналогов реальных приложений для функциональной верификации микропроцессора.

Получение требуемых фрагментов из всей трассы продолжительных тестов возможно с использованием механизма (рассматривался в [1]) срезки-восстановления состояния модели микропроцессора, получаемого при запуске эталонного эмулятора. При использовании этого механизма вся последовательность команд теста разбивается на множество подпоследовательностей, которые выполняются параллельно на разных вычислительных устройствах. Этот способ, однако, обладает значительным недостатком – существует вероятность, что полученный фрагмент теста будет содержать нерепрезентативный код. Экспериментально показано, что адаптация больших тестов сводится к обоснованному сокращению всей трассы теста до

небольших, но показательных фрагментов, позволяющих потенциально повысить покрытие.

Второй вариант получения непродолжительных репрезентативных тестов производительности для RTL-модели микропроцессора заключается в выявлении и описании характерных признаков и особенностей реальных вычислительных задач, входящих в состав тестовых пакетов, с последующей генерацией эквивалентных по этим признакам тестов. Существуют подходы [2 – 10] к анализу сходства тестовых программ с последующим синтезом новых эквивалентных наборов тестов, обладающих теми же ключевыми особенностями и в то же время являющихся гораздо меньшими по сравнению с оригинальным набором тестов. Подобные методы направлены на создание бенчмарков (*benchmarks*), то есть тестов производительности, либо на анализ программ с целью оценить верхнюю границу времени выполнения фрагмента кода (worst case execution time, WCET) – это максимальное время, которое требуется для выполнения данного фрагмента кода [11].

Например, в работе [8] предлагается ввести более 50 параметров, по которым можно судить о сложности кода теста. В качестве примера приводятся следующие метрики:

- набор инструкций: инструкции обращения к памяти, инструкции ветвления, целочисленные арифметические операции, операции с плавающей точкой;
- параллелизм на уровне инструкций;
- зависимости по регистрам;
- количество промахов в кэш-память всех уровней на 1000 обращений;
- степень локальности обращений в сегмент памяти;
- точность предсказания переходов инструкций ветвления.

Метрики можно разделить на зависимые от микроархитектуры (например, количество промахов в кэш-память, точность предсказания инструкций ветвления) и независимые от микроархитектуры (инструкции, входящие в тест, возраст зависимостей по регистрам).

В данной работе подобный список метрик предлагается применить для синтеза набора уменьшенных тестов (эквивалентный «экстракт»), который может быть использован для функциональной верификации RTL-модели микропроцессора. Синтез таких уменьшенных тестов имеет смысл при выполнении двух условий: тест воспроизводит характеристики выполнения машинного кода того приложения, на котором он основан (то есть является репрезентативным набором инструкций микропроцессора), и достигает того же результата покрытия за гораздо меньшее время. Уменьшенный тест представляет собой программу, написанную на языке Ассемблер целевой архитектуры МП. В итоге, множество созданных коротких репрезентативных тестов производительности, время выполнения которых на RTL-модели занимает до 10 минут, используется вместо продолжительных тестов, запуск которых не представляется возможным за требуемое время.

Предлагаемая идея позволит использовать преимущества запуска приложений пользователя с точки зрения обнаружения ошибок в микропроцессоре на раннем этапе разработки его RTL-модели (когда стоимость исправления ошибки минимальна). Также, раннее обнаружение микроархитектурных ошибок может быть достигнуто путем включения в состав псевдослучайных тестов эквивалентного «экстракта» пользовательских программ, построенного с помощью рассматриваемых в статье методов.

II. Обзор

Данная работа во многом основывается на идеях, предложенных в работах [2 – 8]. Их основная тематика – создание небольших по объёму, но при этом репрезентативных по своим показателям тестов производительности (бенчмарков) на основе уже существующих программ. Так, в работе [4] описывается технология искусственного синтеза рабочей нагрузки, создаваемой приложением. Она позволяет воспроизвести процесс выполнения реального проприетарного программного обеспечения в виде синтетического бенчмарка, имеющего похожие характеристики рабочей нагрузки и энергопотребления, но представляющего собой отличный от исходного набор динамически исполняемых инструкций. Таким образом, полученная на выходе программа не разглашает структуру закрытого исходного кода первоначального приложения. Для получения такого синтетического "клона" авторы применяют определённые модели обращений к памяти и модель ветвления, позволяющие собрать и затем воспроизвести присущие исходному приложению характеристики, полностью отражающие уровни локальности данных и предсказуемость потока управления программы (работу блока предсказания ветвлений). Особенно важно то, что полученный бенчмарк исполняется значительно быстрее исходного приложения, позволяя сократить время симуляции на потактовых симуляторах. Названная работа развивает концепции работ, опубликованных ранее. В них используется идея генерации синтетического бенчмарка на основе статистического профилирования программы с целью сокращения её времени выполнения, как, например, в [2]. Для описания потока выполнения программы применяется его построение в виде Statistical Flow Graph (SFG) – статистического графа переходов. Каждая вершина данного графа представляет из себя базовый блок программы (набор инструкций или отдельные функции), а каждая грань – возможный переход между такими блоками, что позволяет более точно воспроизвести исходное распределение нагрузки.

SFG отражает, сколько в действительности времени было проведено в отдельных блоках во время исполнения программы. Использование такой структуры в контексте генерации синтетического бенчмарка впервые было предложено в работе [3].

Дальнейшее развитие этого подхода осуществляется в работе [5], где вводится расширение SFG дополнением его информацией о циклах (их непосредственной идентификацией в графе и обозначением количества циклов в каждом) и названное Statistical Flow Graph with Loop Annotation (SFGL) – статистическим графом переходов с аннотацией циклов. Такая структура позволяет конструировать итоговый бенчмарк из множества вложенных циклов, что точнее отражает поведение реальных приложений. Кроме того, результатом описанного в работе процесса является программный код на языке высокого уровня (в данном случае — язык программирования C). Это позволяет использовать такие бенчмарки для исследования и сравнения между собой как микропроцессорных архитектур, так и различных компиляторов.

Во всех из приведённых работ для описания нагрузки, создаваемой программой, используется ряд характерных численных показателей (метрик), непосредственно отражающих её поведение [6]. Первый тип таких характеристик – зависящие от архитектуры микропроцессора (*architecture-dependent*). Они включают в себя количество инструкций на цикл (Instructions Per Cycle, IPC), процент промахов в разные уровни кэш-памяти, вероятность неверного предсказания переходов, и промахи в TLB (Translation Look-aside Buffer). Второй же тип характеристик называется архитектурно-независимыми (*architecture-independent*). В него входят распределение инструкций, параллелизм уровня инструкций (Instruction-Level Parallelism, ILP), средний шаг доступа к памяти (разница между адресами двух последовательных обращений к памяти), предсказуемость ветвлений и другие. Использование архитектурно-независимых метрик позволяет точнее описать различные аспекты поведения программы, так как они релевантны для любой конфигурации и архитектуры.

Работа [7] расширяет идеи этих работ, предлагая метод автоматизированной генерации синтетических бенчмарков на основе многопоточных приложений. Для этого вводятся дополнительные метрики, описывающие межпроцессорное взаимодействие, которое наблюдается в ходе работы исходной программы. Эти метрики включают в себя паттерны доступа к общим данным, число используемых потоков, характеристики синхронизации (например, частота конфликтов при использовании мьютексов и расстояние между блокировкой и освобождением мьютекса).

В работе [8] в дополнение к метрикам из предыдущей работы добавляются паттерны параллелизации приложений, характеризующие шаблоны проектирования исходных многопоточных приложений. Они позволяют более точно воспроизвести схему взаимодействия программных потоков между собой.

Пример использования упомянутых подходов наглядно отражён в работе [9]. Эти подходы применяются для создания тестов производительности на основе бенчмарков, отражающих реальную нагрузку приложений, работающих с «большими данными», и взаимодействующих с современными системами управления базами данных. В результате достигается высокая схожесть по показателям производительности с исходными приложениями. Так, точность воспроизведения количества инструкций на цикл составила в среднем 94.2%, при том, что число инструкций уменьшилось в среднем в 520 раз.

В данной работе, в отличие от вышеназванных, основной целью является не создание эквивалентных по производительности тестов, а создание синтетических аналогов реальных приложений для функциональной верификации микропроцессора. При этом они должны быть такими же эффективными по покрытию, но выполняться значительно быстрее (в 10^3-10^5 раз).

В результате проведенного анализа, по аналогии с работой [10], предполагается выделять «тестовое знание» из пользовательских приложений. Однако, предлагаемый нами подход отличается от получения непосредственного набора возможных тестовых ситуаций из исходной программы. Вместо этого, создаваемый тест должен самостоятельно отражать «тестовое знание», заложенное в существующем приложении. Это, в свою очередь, достигается за счёт воспроизведения получаемым тестом численных характеристик исходного приложения.

III. Описание генератора

Работу генератора тестов можно структурно разделить на три стадии:

- профилирование исходных приложений;
- сокращение графа;
- воспроизведение исходного поведения (генерация теста).

Для анализа приложения и построения по нему графа можно использовать либо листинг программы, либо трассу её исполнения. В случае трассы, в отличие от листинга, мы получаем информацию о реально исполнявшихся в программе инструкций и о том, сколько в действительности времени было проведено в каждом из отдельных участков программы. Поэтому было решено остановиться на данном варианте.

Таким образом, в первой стадии происходит считывание трассы исполнения программы и построение по ней графа переходов. Базовым блоком (вершиной графа) в данном случае является блок

инструкций, заканчивающийся инструкцией перехода (условного или безусловного). Кроме того, собирается и сохраняется информация о считанных инструкциях для каждого блока и количество осуществлённых в трассе переходов для каждого отдельного перехода.

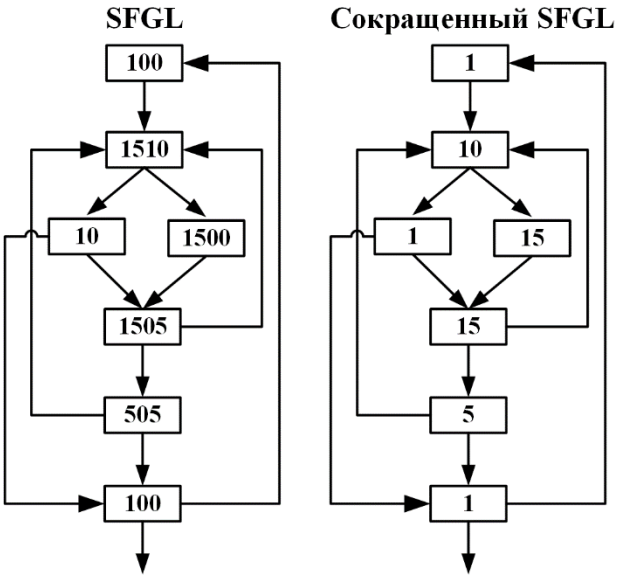

Рис. 1. Оригинальный и сокращенный граф приложения

Сокращение времени исполнения программы достигается следующим образом: полученный граф изменяется уменьшением числа итераций циклов в определённое число раз, определяемое заданным фактором сокращения R. При этом сохраняются грани, количество переходов по которым при таком уменьшении стало бы меньше единицы. Это делается для того, чтобы сохранить все возможные типовые сценарии, которые встречаются при исполнении исходного приложения (рис. 1). В противном случае возможна потеря уникальной тестовой ситуации, возникающей в приложении. Было выделено несколько способов сокращения циклов. Одним из вариантов является сокращение количества итераций каждого цикла в R раз. Однако для глубоко вложенных циклов мы получим итоговое сокращение в R^N раз, что значительно больше изначально задуманного. Также можно сокращать только внешний цикл в R раз, но тогда получается, что вложенные циклы сокращаются слишком слабо. В результате был сделан вывод о необходимости сокращения каждого вложенного цикла в $\sqrt[N]{R}$, где N является уровнем вложенности цикла.

Дальше осуществляется получение линейного массива булевых значений, каждое из которых означает выбор той или иной ветви в определённый момент выполнения программы. Для этого происходит обход графа и на каждом ветвлении случайно выбирается следующая ветвь. При этом вероятность выбора того или иного варианта вычисляется исходя из количества совершённых переходов (то есть из того, сколько раз каждый из данных переходов выполнился) после

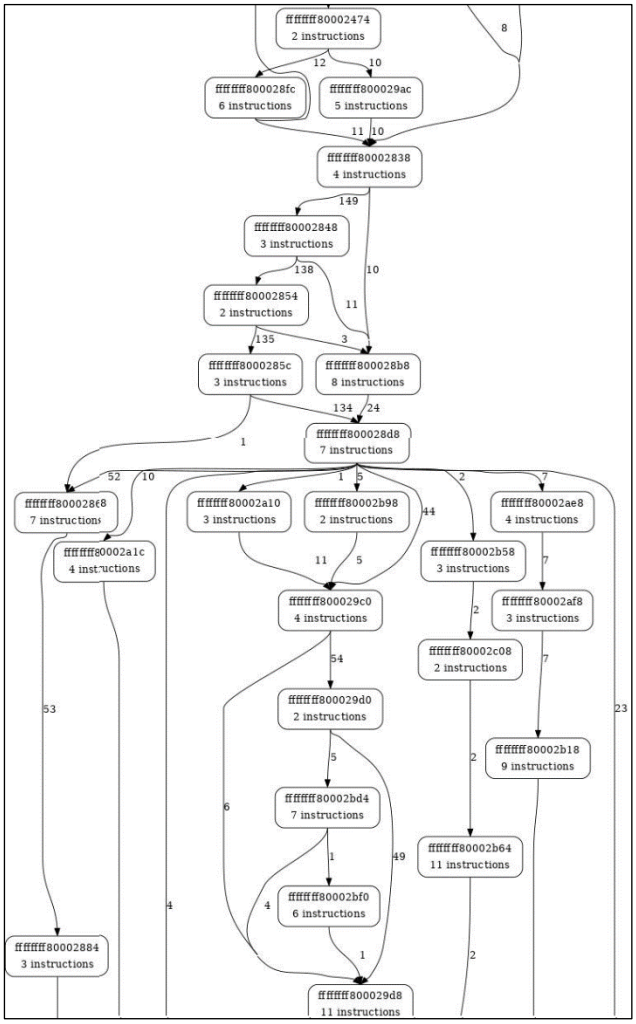

Рис. 2. Фрагмент графа приложения пользователя

сокращения. Пример графа реального приложения пользователя представлен на рис. 2.

Для точного воспроизведения исходного приложения необходимо повторить не только его структуру, но и обратно воспроизвести наполнение каждой вершины инструкциями. При этом код, случайно наполненный инструкциями тех же типов, что и в оригинальном приложении будет показывать схожую производительность со своим эталоном. Однако суть данной работы состоит в сохранении микроархитектурных особенностей оригинального приложения для последующего использования получившихся синтетических клонов в различных стадиях функциональной верификации. Поэтому, исходя из собственного предыдущего опыта создания генераторов случайных тестов [12], а также основываясь на работе [13] о микроархитектурном описании кода, вводятся функциональные метрики (таблица 1). При корректном выборе набора микроархитектурных метрик порождаемый генератором код будет затрагивать те же аспекты архитектуры, что и оригинальное приложение. Это подтверждается сравнением замеров покрытия RTL-

модели оригинального приложения и его синтетического клона.

Таблица 1

Используемые функциональные метрики

Категория	**Метрика**	**Размерность**
Instruction mix	Load	%
	Store	%
	Arithmetic	%
	Logical	%
	FP	%
Instruction locality	Branch predict	%
	Branch locality	Distance
Data locality	Footprint	Address set
	Stride	Sizes set
Dependency	RAW	%, Distance
	WAW	%, Distance
	WAR	%, Distance
Memory	Load-store streams	%, size
	Copy cycles	%, size
Cache	L1D hit, L2 hit	%
	L1D miss, L2 hit	%
	L1D miss, L2 miss	%
	L1I miss	%

На последней стадии производится итоговый обход графа и запись получаемого кода в виде набора ассемблерных инструкций. Они в свою очередь генерируются исходя из наполнения базовых блоков по типам инструкций. Каждое ветвление во время непосредственного исполнения теста сопровождается предварительным считыванием из памяти значения, соответствующего выбору определённой ветви в каждом отдельном случае.

Предложенный метод был апробирован на тестах производительности SPEC2000 INT. Данный пакет программ, изначально разработанный для оценки производительности микропроцессоров и вычислительных систем, в представленной работе был использован как максимально разнообразный набор программ. Для каждого теста был построен граф переходов и, используя представленные ранее метрики, создан короткий синтетический клон. Для оценки качества сгенерированного кода было произведено сравнение покрытия RTL-модели между оригинальными приложениями и их синтетическими клонами (таблица 2). Оригинальные приложения и их синтетический аналоги были запущены на Cadence Simulator со сбором покрытия. Покрытие оценивалось по следующим параметрам:

- Code
  - Block
  - Statement
  - Expression
  - Toggle
- FSM
- Functional
  - Assertions
  - CoverGroup

Таблица 2

Сравнение результатов измерения покрытия между оригинальными приложениями и их синтетическими клонами

SPEC2000 INT TESTS	Coverage, %	
	Оригинал	Клон
164.gzip	68.32	62.66
175.vpr	65.97	59.28
176.gcc	69.41	62.85
181.mcf	60.75	55.92
186.crafty	70.25	62.17
197.parser	70.93	59.26
252.eon	73.29	67.44
254.gap	70.32	63.05
255.vortex	39.96	41.58
256.bzip	70.08	61.15

Как видно из таблицы, синтетические клоны имеют схожее покрытие RTL-кода, а значит, при их выполнении, задействуются те же архитектурные механизмы, что и в оригинальном приложении, при этом время симуляции сокращено в R=100 раз.

IV. Развитие

Основное направление развития рассматриваемого подхода на данный момент заключается в расширении набора отслеживаемых метрик для более детального воспроизведения поведения исходной программы в тесте. На основе этих метрик планируется расширять логику генерации инструкций и операндов для них.

Поскольку созданный инструмент позволяет устанавливать соответствие между функциональным покрытием (набор метрик) и покрытием RTL-кода, то он может быть использован для дальнейших разработок, связанных с поиском максимума покрытия для выбранного приложения.

С помощью управляемой «коррекции» весов распределения инструкций в блоках или вероятностей переходов принципиально можно добиться большего функционального покрытия. Это позволяет дополнить данную архитектуру алгоритмами нейросетей, которые бы вносили подобные изменения и анализировали получаемый результат автоматически. С помощью такого взаимодействия возможно нахождение новых и уникальных тестовых ситуаций, не представленных в исходном приложении.

Заключение

Непрерывно усложняющиеся микропроцессорные системы требуют постоянного развития новых средств тестирования. Предложенный механизм получения коротких синтетических клонов реальных приложений пользователя позволяет начать запускать код, эквивалентный реальным задачам на гораздо более раннем этапе проектирования, когда стоимость исправления найденной ошибки минимальна. В статье подтверждается применимость синтетических аналогов реальных приложений для функциональной верификации моделей микропроцессора.

ЛИТЕРАТУРА

[1] Бобков С.Г., Чибисов П.А. Повышение качества тестирования высокопроизводительных микропроцессоров методами встречного тестирования с анализом функционального тестового покрытия выделенных приложений. // Информационные технологии, №8, 2013, с. 26-33.

[2] L. Eeckhout, K. de Bosschere and H. Neefs. Performance analysis through synthetic trace generation. // 2000 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS (Cat. No.00EX422). 2000. P. 1-6.

[3] L. Eeckhout, R. H. Bell, B. Stougie, K. De Bosschere and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. // Proceedings. 31st Annual International Symposium on Computer Architecture, 2004. 2004. P. 350-361.

[4] Ajay Joshi, Lieven Eeckhout, Robert H. Bell, and Lizy K. John. Distilling the essence of proprietary workloads into miniature benchmarks. // ACM Trans. Archit. Code Optim. 5, 2, Article 10 (August 2008). 2008. P. 1-33.

[5] L. Van Ertvelde and L. Eeckhout. Benchmark synthesis for architecture and compiler exploration. // IEEE International Symposium on Workload Characterization (IISWC'10). 2010. P. 1-11.

[6] K. Hoste and L. Eeckhout. Microarchitecture-Independent Workload Characterization. // IEEE Micro, vol. 27, no. 3. 2007. P. 63-72.

[7] K. Ganesan and L. K. John. Automatic Generation of Miniaturized Synthetic Proxies for Target Applications to Efficiently Design Multicore Processors. // IEEE Transactions on Computers, vol. 63, no. 4. April 2014. P. 833-846

[8]. E. Deniz, A. Sen, B. Kahne and J. Holt. MINIME: Pattern-Aware Multicore Benchmark Synthesizer. // IEEE Transactions on Computers, vol. 64, no. 8. 2015. P. 2239-2252.

[9]. R. Panda and L. K. John. Proxy Benchmarks for Emerging Big-Data Workloads. // 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). 2017. P. 105-116.

[10] W. Chen, L. Wang, J. Bhadra and M. Abadir. Simulation knowledge extraction and reuse in constrained random processor verification. // 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). 2013. P. 1-6.

[11] Grades, Zur & Ingenieurwissenschaften, Doktor & I, Der & Theiling, Henrik & Prof, Vorsitzender & Weikum, Dr & Prof, Gutachter & Wilhelm, Reinhard & Dr, Prof & Ganzinger, Harald. (2003). Control Flow Graphs for Real-Time System Analysis: Reconstruction from Binary Executables and Usage in ILP-Based Path Analysis.

[12] Смирнов А.В., Чибисов П.А. Генератор тестов для проверки когерентности кэш-памятей многоядерных микропроцессоров (ristretto) // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2018. Вып. 2. С. 31-38. doi:10.31114/2078-7707-2018-2-31-38

[13] Гревцев Н.А., Хисамбеев И.Ш., Чибисов П.А. Исследование способов повышения эффективности стохастического тестирования моделей микропроцессоров // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 2. С. 8-15.

User Applications' Synthetic Clones Generation for Functional Verification

N.A. Grevtsev, A.A. Krasnyuk, D.O. Orlov, P.A. Chibisov

SRISA RAS, Moscow, denorl2009@gmail.com

***Abstract* — Low speed of RTL-level code simulation is the main reason behind the practical inability of a full-fledged startup of an operating system and end user applications under OS control on RTL simulators. This makes it impossible to guarantee the lack of critical errors in a taped-out microprocessor VLSI. An operating system startup alone on a target working frequency could allow us to find the majority of the remaining errors before they escape into the production stage. This paper presents a technique of user applications synthetic clone creation which may be used for the functional verification of the target microprocessor's RTL model. Synthesis of reduced test clones may be considered practical under two conditions. The test must repeat the given machine code execution characteristics of a source application (which means that it is a representative set of processor instructions). It must also attain the same coverage results in much less time. An amount of such short and representative tests, which can run on the RTL model in less than 10 minutes, might be used instead of lengthy original programs, which cannot be executed in a given time. The presented approach allows us to utilize the benefits of user application execution from a point of error detection at early stage of microprocessor's RTL model, when the cost of its correction is minimal. Early detection of micro-architectural errors may also be achieved by inclusion of an equivalent "extract" of user applications, obtained by presented techniques, into the pseudo random tests.**

***Keywords* — functional verification, synthetic benchmarks, coverage metrics, performance tests, control flow graph, guided test generation, cyclomatic complexity of a program, Synthetic Benchmark.**

REFERENCES

[1] Bobkov S.G., Chibisov P.A. Povyshenie kachestva testirovanija vysokoproizvoditel'nyh mikroprocessorov metodami vstrechnogo testirovanija s analizom funkcional'nogo testovogo pokrytija vydelennyh prilozhenij. // Informacionnye tehnologii, №8, 2013, s. 26-33. (in Russian).

[2] L. Eeckhout, K. de Bosschere and H. Neefs. Performance analysis through synthetic trace generation. // 2000 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS (Cat. No.00EX422). 2000. P. 1-6.

[3] L. Eeckhout, R. H. Bell, B. Stougie, K. De Bosschere and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. // Proceedings. 31st Annual International Symposium on Computer Architecture, 2004. 2004. P. 350-361.

[4] Ajay Joshi, Lieven Eeckhout, Robert H. Bell, and Lizy K. John. Distilling the essence of proprietary workloads into miniature benchmarks. // ACM Trans. Archit. Code Optim. 5, 2, Article 10 (August 2008). 2008. P. 1-33.

[5] L. Van Ertvelde and L. Eeckhout. Benchmark synthesis for architecture and compiler exploration. // IEEE International Symposium on Workload Characterization (IISWC'10). 2010. P. 1-11.

[6] K. Hoste and L. Eeckhout. Microarchitecture-Independent Workload Characterization. // IEEE Micro, vol. 27, no. 3. 2007. P. 63-72.

[7] K. Ganesan and L. K. John. Automatic Generation of Miniaturized Synthetic Proxies for Target Applications to Efficiently Design Multicore Processors. // IEEE Transactions on Computers, vol. 63, no. 4. April 2014. P. 833-846

[8]. E. Deniz, A. Sen, B. Kahne and J. Holt. MINIME: Pattern-Aware Multicore Benchmark Synthesizer. // IEEE Transactions on Computers, vol. 64, no. 8. 2015. P. 2239-2252.

[9]. R. Panda and L. K. John. Proxy Benchmarks for Emerging Big-Data Workloads. // 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). 2017. P. 105-116.

[10] W. Chen, L. Wang, J. Bhadra and M. Abadir. Simulation knowledge extraction and reuse in constrained random processor verification. // 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). 2013. P. 1-6.

[11] Grades, Zur & Ingenieurwissenschaften, Doktor & I, Der & Theiling, Henrik & Prof, Vorsitzender & Weikum, Dr & Prof, Gutachter & Wilhelm, Reinhard & Dr, Prof & Ganzinger, Harald. (2003). Control Flow Graphs for Real-Time System Analysis: Reconstruction from Binary Executables and Usage in ILP-Based Path Analysis.

[12] Smirnov A.V., Chibisov P.A. Random Test Generator for Multicore Microprocessor Cache Coherence Verification (Ristretto) // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2018. Issue 2. P. 31-38. doi:10.31114/2078-7707-2018-2-31-38

[13] Grevtsev N.A., Khisambeev I.Sh., Chibisov P.A. Methods to improve efficiency of microprocessor model stochastic tests // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2016. Part 2. P. 8-15.