**ISSN 2413–9599 (Print)**
**ISSN 2541–870X (Online)**

**www.radioprom.org**

DOI: 10.21778/2413-9599-2019-29-2-17-27
УДК 004.414.23

# Разработка потактового симулятора подсистемы памяти процессорного ядра «Эльбрус»

**Д. В. Знаменский[1], В. Н. Куцевол[1]**

[1] АО «МЦСТ», Москва, Россия

Возрастающая сложность микропроцессоров, а также замедление прогресса микроэлектронной технологии делают дальнейшее повышение производительности все более затруднительным. В связи с этим актуальность оценки производительности перспективных микропроцессоров с помощью потактового моделирования перед их реализацией в кремнии увеличивается. В статье изложен подход к реализации потактового симулятора подсистемы памяти процессорного ядра для архитектуры «Эльбрус», управляемого существующим функциональным симулятором этой архитектуры. Рассмотрена методика оценки точности потактового симулятора в сравнении с моделированием RTL-описания перспективного микропроцессора. Приведены данные по быстродействию потактового симулятора и основные техники, позволившие добиться приемлемой производительности. Даны полученные с помощью потактового симулятора предварительные оценки влияния на производительность некоторых изменений в перспективном процессорном ядре, включая задержку доступа в кэш и аппаратную поддержку виртуализации. Эти оценки важны для принятия архитектурных решений при проектировании перспективных процессоров архитектуры «Эльбрус».

***Ключевые слова:*** моделирование, микропроцессор, потактовый симулятор, функциональный симулятор, архитектура «Эльбрус», кэш-память, виртуализация

*Для цитирования:*
Знаменский Д. В., Куцевол В. Н. Разработка потактового симулятора подсистемы памяти процессорного ядра «Эльбрус» // Радиопромышленность. 2019. Т. 29, № 2. С. 17–27. DOI: 10.21778/2413-9599-2019-29-2-17-27

# Development of a cycle-accurate simulator of the Elbrus processor core memory subsystem

**D.V. Znamenskiy[1], V.N. Kutsevol[1]**

[1] MCST JSC, Moscow, Russia

Increasing complexity of modern microprocessors, combined with semiconductor technology progress slowdown, make a further increase in performance more difficult. Under these circumstances, the relevance of the performance estimations of prospective microprocessors by dint of cycle-accurate simulation prior to their production in silicon is of growing importance. The approach to implementation of cycle-accurate simulator of core memory subsystem for Elbrus architecture, controlled by the existing functional simulator of this architecture, is presented herein. The method for validation of a cycle-accurate simulator by comparison with modeling of the RTL description of the prospective microprocessor is considered. The data on the speed of the cycle-accurate simulator and the main optimization methods, which were used to achieve acceptable performance, are presented. The preliminary estimates of the impact on the performance of some changes in the prospective processor core, including the cache access latency and hardware support for virtualization, obtained with the help of the cycle-accurate simulator are given. These assessments are important for making architectural decisions when designing the prospective Elbrus architecture processors.

***Keywords:*** simulation, microprocessor, cycle-accurate simulator, functional simulator, Elbrus architecture, cache memory, virtualization

*For citation*:
Znamenskiy D.V., Kutsevol V.N. Development of cycle-accurate simulator of the Elbrus processor core memory subsystem. Radiopromyshlennost, 2019, vol. 29, no. 2, pp. 17–27 (In Russian). DOI: 10.21778/2413-9599-2019-29-2-17-27

**Введение**

Повышение производительности микропроцессоров в рамках существующей фон неймановской парадигмы требует все большего увеличения используемой площади кристалла и усложнения микроархитектуры. В то же время прогресс полупроводниковой технологии замедляется в силу объективных физических и экономических причин, и для поддержания тренда повышения производительности требуются все более тонкие изменения микроархитектуры. Эта задача становится более трудоемкой, так как возрастающая сложность микропроцессорных ядер, в свою очередь, усложняет теоретическую оценку качества этих изменений и снижает ее точность. Существенно возрастающая при этом актуальность моделирования микроархитектуры перед ее реализацией в кремнии обусловила разработку представленного в данной статье потактового симулятора (ПС) подсистемы памяти архитектуры «Эльбрус», обеспечивающего, с одной стороны, необходимую точность оценок, а с другой – обладающего достаточной производительностью. Изложенный подход к построению ПС, управляемого существующим функциональным симулятором (ФС), позволяет упростить проектирование путем повторного использования части логики ФС. В статье рассмотрены методика получения и результаты оценки точности ПС путем сравнения его динамики с динамикой RTL (Register Transfer Level)-модели реального процессора. Представлен обзор основных техник, позволивших достичь уровня быстродействия ПС, достаточного для моделирования на длительных тестовых нагрузках. Приведены полученные на ПС оценки производительности микроархитектурных изменений в перспективных процессорах семейства «Эльбрус». Кроме того, изложена методика получения оценок и даны предварительные оценки производительности вводимых в архитектуру средств аппаратной поддержки виртуализации.

**Постановка задачи**

Для таких распространенных процессорных архитектур, как x86, создан целый ряд функциональных и потактовых симуляторов [1]. Создание ФС для архитектуры «Эльбрус», поддерживающего потактовый режим, описано в [2], где основное внимание уделялось моделированию конвейерных задержек и блокировок по зависимостям между операциями. Авторы настоящей статьи сосредоточились на моделировании подсистемы памяти процессорного ядра, руководствуясь следующими соображениями.

«Эльбрус» – архитектура широкого командного слова (VLIW) со статическим планированием вычислений. При этом в существующих машинах выдача команд на исполнение осуществляется в программном порядке (in-order). Как правило, компилятор планирует вычисления достаточно точно: в случае

арифметико-логических операций задержки исполнения известны и для большинства из них (хотя и не для всех) фиксированы, что позволяет компилятору, насколько возможно, скрывать задержки от источника до потребителя результата. Аналогичное утверждение верно и для операций передачи управления с явной командой подготовки. Из этих предпосылок следует вывод, что основным источником задержек исполнения для процессорного ядра архитектуры «Эльбрус», имеющим недетерминированное поведение, является подсистема памяти. Таким образом, было решено создать именно ПС подсистемы памяти для оценки поведения системы в целом. Аналогичные работы по моделированию «нижней» части процессора (кэш последнего уровня, сеть межсоединений) в многоядерной конфигурации проводились и описаны в [3, 4].

Перед авторами настоящей статьи стояла задача разработать ПС подсистемы памяти ядра архитектуры «Эльбрус» со следующими свойствами:

- точное микроархитектурное моделирование потерь при обращении к памяти (блокировки по данным, подкачке кода, ресурсные блокировки);
- возможность инкрементально наращивать точность моделирования (модульный принцип);
- возможность оценки точности моделирования в сравнении с RTL-описанием реального процессора;
- производительность, достаточная для моделирования исполнения типовых задач (загрузка ядра Linux, бинарый компилятор, пакет задач SPEC, Coremark и др.).

В качестве основы этого инструмента был выбран ФС, непосредственно исполняющий двоичные коды архитектуры «Эльбрус». Информация об исполненных им широких командах (ШК) целевой архитектуры передается в ПС для точного моделирования задержек в соответствии с микроархитектурой процессорного ядра. Такая схема, с одной стороны, позволила упростить ПС, который освобождается от функции определения траектории исполнения программы с точки зрения точного архитектурного состояния (содержимое регистров процессора, памяти и т.д.), а с другой – дала возможность напрямую использовать информацию о динамике исполнения программы и архитектурном состоянии машины, сформированную ФС.

**Взаимодействие функционального и потактового симуляторов**

Архитектура связки ФС и ПС представлена на рис. 1.

Функциональный симулятор в цикле осуществляет выборку, декодирование и выполнение широких команд. Информация об их исполнении, формируемая ФС и передаваемая на вход ПС, содержит две группы полей:

1) содержимое ШК в расшифрованном и дополненном виде: указатель адреса команды, операции обращения в память, номера регистров-операндов и регистров назначения, операции передачи управления и пр.;
2) данные об архитектурной динамике: содержимое таблиц страниц (ТС), информация о прерываниях и пр.

Вторая группа полей критически важна для выбранной схемы симуляции. Функциональный симулятор при операциях обращения в память производит трансляцию физического адреса как для

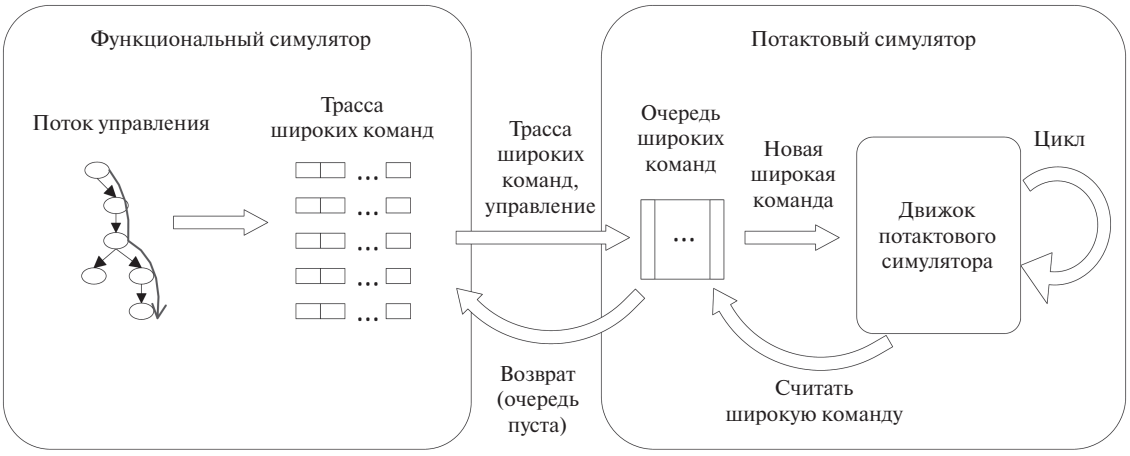


Рисунок 1. Взаимодействие функционального и потактового симуляторов
Figure 1. The interaction between functional and cycle-accurate simulators

адресов инструкций, так и для адресов данных. При первой трансляции ФС проходит все уровни ТС и сохраняет в TLB (Translation Lookaside Buffer) данные о дескрипторах страниц в цепочке трансляций (PTE chain) вместе с дескриптором конечной страницы. При следующих запросах в оттранслированные страницы ФС передает сохраненную цепочку трансляций ФС. Использование этой информации позволяет снять с ПС задачу непосредственно трансляции виртуальных адресов и оставить лишь функцию моделирования задержек трансляции (обращение к TLB и кэшам). Это же касается и прерываний: ПС не формирует прерывания в динамике, а использует информацию о прерываниях от ФС для моделирования задержек.

Существенная особенность данной схемы – то, что с точки зрения памяти ПС не работает с данными – содержимым физической памяти и регистров процессора (за исключением некоторых настроек), но моделирует динамику обработки в зависимости от адреса обращения в память.

**Архитектура потактового симулятора**

Архитектура ПС представлена на рис. 2.

Движок ПС можно представить в виде синхронного с трассой ШК цикла конвейера, каждая итерация которого соответствует одному машинному такту. На каждой итерации выполняется код модулей подсистемы памяти ядра (кэши, буферы и пр.). Кроме того, вычисляются некоторые глобальные (уровня конвейера) блокировки, например по неготовности операнда ШК. Сами модули могут быть либо напрямую привязанными к конвейеру (модули первого типа), либо асинхронными относительно конвейера (модули второго типа) (в этом случае их работа привязана к модулям первого типа, а к конвейеру – только косвенным образом).

Работа модулей обоих типов состоит в обработке событий, которые могут относиться к одному из трех классов:

1) конвейерные (синхронные) – события, поступающие напрямую из цикла конвейера: операции ШК (чтения и записи) и считывания кода. Глобальные блокировки приостанавливают синхронный поток команд и обработку соответствующих конвейерных событий;
2) спланированные (отложенные) – вторичные события, формируемые модулями ПС, которые обрабатываются при наступлении определенного такта (с известными задержками), например выдача результата просмотра кэша;
3) неотложные (приоритетные) – вторичные события, требующие обработки начиная с момента их создания. Обработка события происходит сразу же после того, как исчезает блокирующее ее условие, например занятость требуемого канала обращения к памяти.

Модули, входящие в состав ПС, соответствуют структуре моделируемого процессорного ядра: кэш команд (IB) и буфер страничной трансляции кода (ITLB), кэш данных первого уровня (L1$), кэши страничной трансляции (TLB, кэши устройства поиска по ТС), кэши второго и третьего уровней (L2$ и L3$) и «заглушка» оперативной памяти. Основным управляющим блоком является конвейер. Модули IB, ITLB, L1$ и TLB можно логически объединить

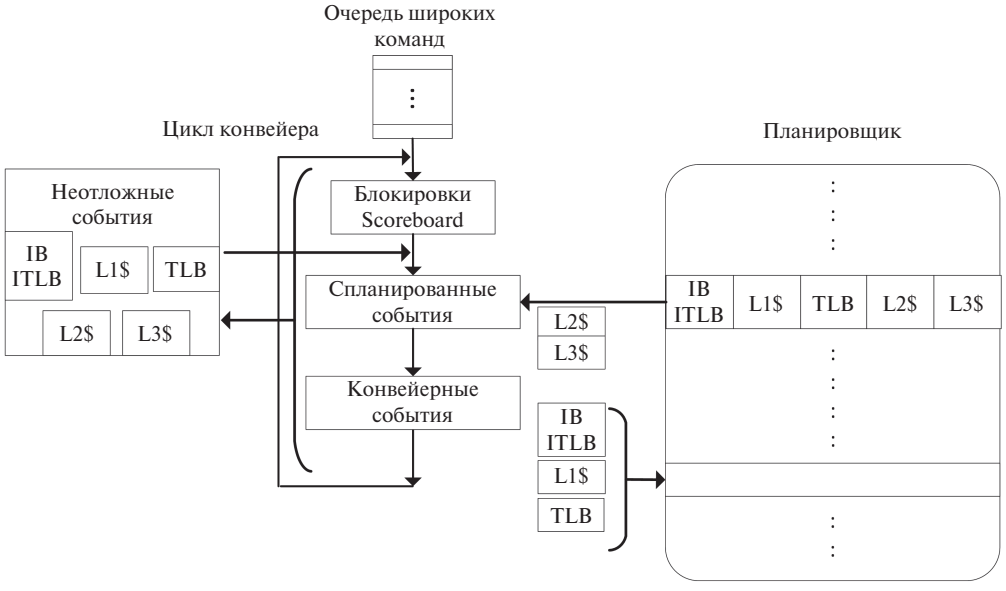


Рисунок 2. Архитектура потактового симулятора
Figure 2. Architecture of the cycle-accurate simulator

в первую группу, так как их работа привязана к основному потоку команд (за исключением механики подкачки массивов) и они управляются непосредственно через основной цикл конвейера. Это соответствует существующей in-order реализации архитектуры «Эльбрус» со статическим планированием кода.

События первого класса обрабатываются только описанной выше группой модулей. Модули L2$ и L3$ (вторая группа) отвязаны от конвейера и управляются запросами от привязанных к конвейеру блоков (событиями второго и третьего классов), т.е. работают асинхронно относительно основного программного потока. Каждый из модулей с разной степенью точности воспроизводит алгоритмы и внутреннюю структуру соответствующего аппаратного узла, например для кэшей это адресация, ассоциативность, внутренние буферы, блокировки и пр. Блок Scoreboard отслеживает зависимости по данным между ШК, что позволяет воспроизводить в симуляторе блокировки по данным.

Модульный подход позволяет постепенно увеличивать точность ПС, добавляя модели структур реальной системы и углубляя существующие модели. При этом по мере наращивания глубины моделирования неизбежно снижается скорость ПС. Для получения оптимального решения были приняты следующие принципы:

- приоритет по точности имеют механизмы, критичные с точки зрения производительности системы;
- применяется упрощенное моделирование либо отказ от моделирования слабо влияющих на производительность механизмов;
- точность моделирования снижается от нижних уровней иерархии к верхним (IB/ITLB, TLB, L1$ → L2$ → L3$).

**Двухуровневая трансляция**

Одним из применений ПС является оценка производительности двухуровневой трансляции адреса, добавленной в рамках аппаратной поддержки виртуализации в новой версии архитектуры «Эльбрус» [5]. Такой механизм позволяет на аппаратном уровне транслировать физические адреса гостя (виртуальной машины) в физические адреса системы. При этом гостевая операционная система контролирует трансляцию гостевых виртуальных адресов в гостевые физические адреса с помощью имеющейся ТС, а гипервизор – трансляцию гостевых физических адресов в системные физические адреса с помощью специальной ТС [6, 7].

Оценка скорости данного механизма осложнялась тем, что на момент создания ПС программное обеспечение для виртуализации (прежде всего, нативный гипервизор) не было готово, а поддержка виртуализации в ФС не была до конца реализована. В связи с этим было предложено моделирование гостевого режима с добавлением слоя адресной трансляции через программно сгенерированную ТС гипервизора (рис. 3).

Особенности предложенной техники заключаются в следующем:

- режим работы ФС считается гостевым – соответственно, и физические, и виртуальные адреса в трассе ШК становятся гостевыми;
- в ПС вводится аппаратная поддержка второго уровня трансляции – ТС гипервизора, модули соответствующих кэшей и алгоритмы поиска. Добавленная логика используется для трансляции гостевых виртуальных и физических адресов в системные физические адреса;
- адреса из трассы ШК, трактуемые как гостевые, пропускаются через второй уровень трансляции согласно сгенерированной ТС гипервизора, описывающей карту физической памяти реальной машины, на которой работает гостевой ФС. Таблица страниц формируется в двух вариантах с точки зрения используемого размера страниц системной памяти – 4 Кбайт либо 2 Мбайт. Через ТС гипервизора гостю выделяется физическая

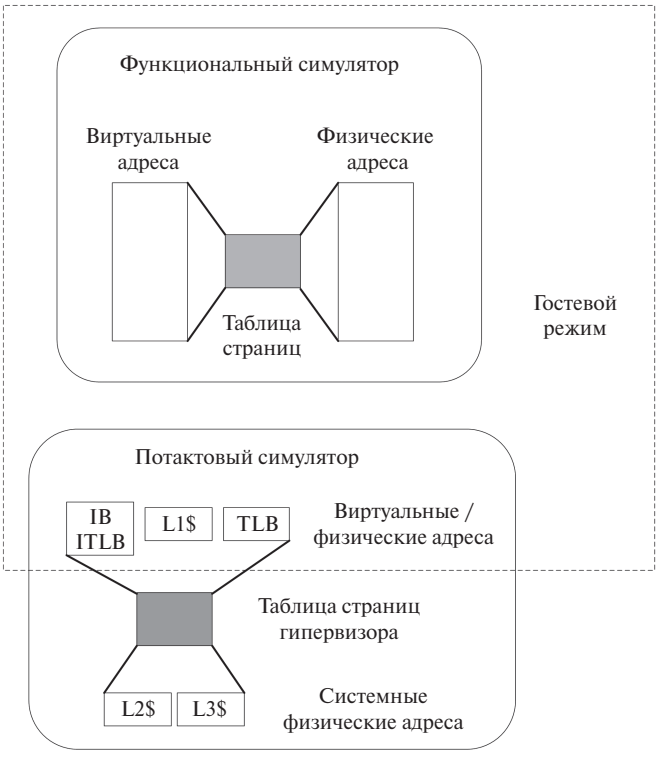


Рисунок 3. Двухуровневая адресная трансляция в потактовом симуляторе
Figure 3. Two-level address translation in the cycle-accurate simulator

память (области загрузчика, ввода-вывода и 1 Гбайт оперативной памяти), выделенные страницы плотно упаковываются в сформированной системной физической памяти;

- единственная структура, требующая непосредственного размещения данных в созданной сгенерированной таблице физической памяти, – это сама сгенерированная ТС. Потактовый симулятор использует модель системной физической памяти ТС, которая хранит данные только для адресов сгенерированной ТС гипервизора. При промахе в трансляционных кэшах ядра происходит обращение по данным к системной памяти ТС для получения содержимого соответствующей PTE.

**Точность и валидация**

Ввиду наличия доступа к RTL целевого процессорного ядра валидация ПС проводилась путем сравнения его динамики с динамикой RTL на некотором наборе тестов. В качестве базы для валидации использовался законченный RTL процессора предыдущего поколения; для оценок перспективного ядра соответствующие доработки в ПС включались инкрементально. Для сравнения были использованы два класса метрик:

1) поведенческие (событийные) метрики – количество внутренних событий определенного типа для каждого блока микроархитектуры и соответствующего ему модуля ПС, например обращения в L1$, попадания в L2$, количество запросов поиска по ТС, количество аппаратных операций установки accessed/dirty битов в ТС и др.;
2) временные метрики, основная часть которых – количество тактов работы машины на заданном тесте. При разборе отклонений по точности использовались и другие временные метрики, например количество тактов блокировки конвейера.

В исходный RTL процессорного ядра была добавлена логика подсчета большого количества типов внутренних событий. Заметим, что конфигурация RTL процессора была, насколько возможно, приведена в соответствие конфигурации ПС. Например, включалось только одно ядро, задержки доступа к оперативной памяти DDR4 сглаживались – аппаратный механизм периодического обновления динамических запоминающих ячеек (REFRESH) был выключен и т.д.

Сбор данных по метрикам проводился при выполнении набора из ~100 специальных тестов различной длительности (от нескольких тысяч до миллионов тактов) на RTL-симуляторе и на ПС, ориентированных в основном на проверку корректности работы механизмов адресной трансляции. Поведенческие метрики использовались для первичной отладки ПС, а временные – для его окончательной валидации. На основании анализа обнаруженных отклонений по метрикам дорабатывались ПС и ФС, и цикл валидации повторялся заново. За ошибку ПС было принято относительное отклонение количества тактов работы ПС от реального RTL на конкретной задаче из набора тестов (в процентах). Достигнутое среднее геометрическое отклонение по всему набору тестов составило около 2%, что можно считать достаточно хорошим результатом [8–11].

В то же время в существующем ПС не все архитектурные механизмы воспроизведены точно, например асинхронные обращения в память (для чистки стека пользователя или подкачки массивов) реализованы упрощенно, что приводит к увеличению ошибки на некоторых классах задач до 30%. В следующих версиях ПС предполагается более точная реализация этих механизмов.

**Производительность потактового и функционального симуляторов**

Скорость работы функционального и потактового симуляторов оценивалась на машине Intel Core i7–3770 @ 3.40 GHz с памятью 16 Гбайт DDR3 под управлением ОС Linux. Быстродействие ФС без ПС можно оценить по его логической скорости (тиков/с): от 500 КГц до 5 МГц при средней скорости порядка 2 МГц в зависимости от исполняемого кода. Быстродействие связки ФС и ПС можно измерять как в тиках ФС, так и в тактах ПС в единицу времени. Средняя логическая скорость связки, измеряемая в тиках в секунду, при тех же условиях составила порядка 500 КГц, а в тактах в секунду – в два раза выше и более, при этом в зависимости от исполняемого кода она может достаточно сильно варьироваться.

Проблема снижения скорости ПС относительно ФС известна [12], и чем точнее потактовая модель, тем заметнее это снижение. Один из путей решения этой проблемы – симуляция с выборкой (sampling simulation) [13–15]. Авторы настоящей статьи выбрали простейший вариант этой методики – непрерывную работу ФС с периодическим запуском ПС (рис. 4).

Временной интервал работы ПС делится на две части: интервал «прогрева» микроархитектурного состояния, в течение которого учет показателей производительности не ведется, и непосредственно измеряемый интервал (квант). В настройках ФС при запуске можно задавать в тиках следующие параметры: периодичность выборки ПС, длительность интервалов «прогрева» и измеряемого кванта ПС. В дальнейшем предполагается усовершенствовать

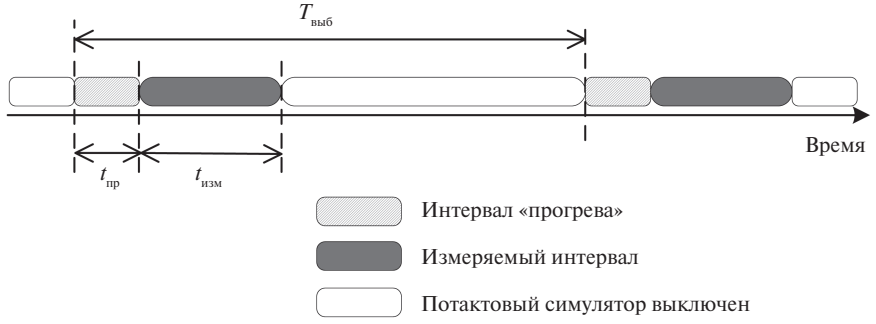


Рисунок 4. Режим симуляции с выборкой: $T_{выб}$ – период выборки; $t_{пр}$ – длительность интервала «прогрева»; $t_{изм}$ – длительность измеряемого интервала
Figure 4. Sampling simulation mode: $T_{выб}$ – sampling period; $t_{пр}$ – warmup period duration; $t_{изм}$ – measured period duration

выборку квантов ПС, например использовать адаптивную методику «прогрева» для задач пакета SPEC [16].

Помимо режима работы ПС с выборкой, в целях ускорения моделирования, для ПС была адаптирована существующая в ФС функциональность сохранения/восстановления состояния симулятора. Эта техника позволяет для запуска конкретной задачи не дожидаться загрузки ОС на связке ФС и ПС, а восстановить ФС и ПС в состоянии уже загруженной ОС, предварительно сохранив его в контрольной точке.

Наряду с описанными мерами применялись различные алгоритмические оптимизации ФС и ПС. Например, для работы в системе с ПС был адаптирован программный кэш широких команд (ICache) ФС – ассоциативный контейнер, предназначенный для хранения интерпретированных команд и доступа к ним по виртуальному адресу команды [12]. Этот механизм существенно сокращает время, необходимое на выборку и декодирование команд: его внедрение в систему увеличило производительность более чем в два раза. Кроме того, в коде ПС вызовы динамического выделения и освобождения памяти были по возможности сведены к минимуму алгоритмическим путем и за счет использования соответствующих структур данных.

**Применение потактового симулятора**

На разработанном ПС были получены предварительные оценки следующих доработок (архитектурных изменений) процессорного ядра «Эльбрус»:

- увеличения задержки доступа в L1$ и длительности конвейерной блокировки («вертушки»);
- добавления двухуровневой трансляции.

Для достижения тактовой частоты 2 ГГц в новой, шестой версии ядра «Эльбрус» потребовалось увеличить задержку доступа в L1$. Это изменение связано с механизмом конвейерной блокировки («вертушки»): при неготовности результата чтения из памяти конвейер блокируется на фиксированное число тактов, по истечении которого конвейерная «вертушка» «поворачивается», и готовность операнда проверяется повторно. Таким образом, оптимальная длительность блокировки коррелирует с задержкой обращения в кэш. Кроме того, длительность «вертушки» может быть увеличена в целях дальнейшего повышения тактовой частоты.

Для ядра шестой версии был выбран вариант увеличения задержки L1$ на один такт без сдвига «вертушки». Этот вариант, а также исходная конфигурация ядра предыдущей, пятой версии и несколько других конфигураций для новой версии ядра были оценены с помощью ПС на загрузке нативного ядра Linux длительностью 500 млн широких команд.

В табл. 1. приведены результаты, полученные на одной из ранних версий ПС для шести различных конфигураций. Модуль L3$ был выключен, задержка доступа к памяти составляла 120 тактов.

В табл. 2. приведены аналогичные результаты на актуальной версии ПС с включенным модулем L3$ и задержкой доступа в память 120 тактов. Несмотря на добавление модуля L3$, длительность моделирования в тактах симулятора имеет возросшую величину в силу уточнения самого ПС.

Из данных табл. 1 и 2 следует, что по параметру замедления конфигурации внутри пар C, D и E, F различаются мало, в то же время разница между самими парами составляет 5–6%. Таким образом, сдвиг «вертушки» сам по себе сильнее влияет на производительность, чем увеличение времени доступа в L1$. Выбранная для ядра шестой версии конфигурация B (увеличенная задержка доступа в L1$ на один такт) на старой версии ПС

Таблица 1. Загрузка ядра Linux. Задержки L1$ и глубина «вертушки» (начальная версия потактового симулятора, L3$ выключен)
Table 1. Linux kernel boot. L1$ latency and the depth of the pipeline replay loop (the initial version of the cycle-accurate simulator, L3$ is off)

| Конфигурация / Configuration | Описание конфигурации / Configuration description | Такты, млн / Cycles, mln | Замедление, % / Slowdown, % | Такты блокировки, млн / Stall cycles, mln | Рост тактов блокировки, % / Stall cycles increase, % | Попадания в кэш, % / Cache hits, % | |
|---|---|---|---|---|---|---|---|
| | | | | | | L1$ | L2$ |
| A (ядро версии 5) | L1$_lat=3, pipe_replay=4 | 1070 | – | 477 | – | 83,8 | 76,4 |
| B (ядро версии 6) | L1$_lat=4, pipe_replay=4 | 1145 | 7,0 | 533 | 11,7 | 83,7 | 76,4 |
| C | L1$_lat=4, pipe_replay=5 | 1206 | 12,7 | 603 | 26,4 | 82,7 | 77,1 |
| D | L1$_lat=5, pipe_replay=5 | 1220 | 14,0 | 614 | 28,7 | 82,7 | 77,1 |
| E | L1$_lat=5, pipe_replay=6 | 1279 | 19,5 | 679 | 42,4 | 83,7 | 76,4 |
| F | L1$_lat=6, pipe_replay=6 | 1280 | 19,6 | 680 | 42,6 | 83,7 | 76,4 |

Таблица 2. Загрузка ядра Linux. Задержки L1$ и глубина «вертушки» (последняя версия потактового симулятора, L3$ включен)
Table 2. Linux kernel boot. L1$ latency and the depth of the pipeline replay loop (the last version of the cycle-accurate simulator, L3$ is on)

| Конфигурация / Configuration | Описание конфигурации / Configuration description | Такты, млн / Cycles, mln | Замедление, % / Slowdown,% | Такты блокировки, млн / Stall cycles, mln | Рост тактов блокировки, % / Stall cycles increase, % | Попадания в кэш, % / Cache hits, % | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | L1$ | L2$ | L3$ |
| A (ядро версии 5) | L1$_lat=3, pipe_replay=4 | 1192 | – | 427 | – | 83,9 | 76,6 | 64,9 |
| B (ядро версии 6) | L1$_lat=4, pipe_replay=4 | 1205 | 1,1 | 437 | 2,3 | 83,9 | 76,6 | 64,9 |
| C | L1$_lat=4, pipe_replay=5 | 1265 | 6,1 | 504 | 18,0 | 83,9 | 76,6 | 64,9 |
| D | L1$_lat=5, pipe_replay=5 | 1266 | 6,2 | 506 | 18,5 | 83,9 | 76,6 | 64,9 |
| E | L1$_lat=5, pipe_replay=6 | 1326 | 11,3 | 571 | 33,7 | 83,9 | 76,6 | 64,9 |
| F | L1$_lat=6, pipe_replay=6 | 1328 | 11,4 | 572 | 34,0 | 83,9 | 76,6 | 64,9 |

с выключенным L3$ отличается от базовой конфигурации A на 7%, а при включенном L3$ – всего на 1%.

Моделирование двухуровневой трансляции проводилось по описанной методологии на основе сгенерированной ТС гипервизора с использованием одной из ранних версий ПС без модуля L3$ с задержкой доступа в память величиной 120 тактов. В качестве нагрузки также была выбрана загрузка нативного ядра Linux. Проводилась оценка трех конфигураций: нативная загрузка Linux, а также загрузка Linux в гостевом режиме через сгенерированную ТС со страницами размерами 4 Кбайт и 2 Мбайт. Результаты представлены в табл. 3.

Из приведенных в табл. 3 данных следует, что быстродействие виртуализованной системы на загрузке Linux заметно выше при выделении системной памяти большими страницами (2 Мбайт), чем страницами меньшего размера (4 Кбайт): в первом случае деградация всего 13%, а во втором – 140%. Такое значительное различие объясняется большой долей гостевых обращений по физической памяти на данной задаче (загрузке Linux). В моделируемой схеме аппаратуры при выключенной у гостя трансляции виртуальных адресов основной TLB используется для кэширования трансляций гостевых физических адресов в системные, при этом для страниц размером 2 Мбайт это кэширование существенно эффективнее, чем для страниц размером 4 Кбайт. Кроме того, для страниц размером 2 Мбайт требуется меньше запросов поиска. Ожидается, что включение L3$ ускорит двухуровневую трансляцию в целом и уменьшит разницу между двумя вариантами ТС.

**Выводы**

Разработан ПС подсистемы памяти ядра «Эльбрус», управляемый существующим ФС этой

Таблица 3. Загрузка ядра Linux. Нативный режим и двухуровневая трансляция (L3$ выключен)
Table 3. Linux kernel boot. Native mode and two-level translation (L3$ is off)

| Метрика / Metrics | Конфигурация / Configuration | | |
|---|---|---|---|
| | нативная / native | гостевой режим, страницы 4 Кбайт / guest mode, 4 KB pages | гостевой режим, страницы 2 Мбайт / guest mode, 2 MB pages |
| Такты, млн / Cycles, mln | 1059 | 2547 | 1192 |
| Замедление,% / Slowdown,% | – | 140 | 13 |
| Попадания в IB,% / IB hits,% | 99,5 | 99,5 | 99,5 |
| Попадания в L1$,% / L1$ hits,% | 84,4 | 83,4 | 84,2 |
| Попадания в L2$,% / L2$ hits,% | 75,1 | 86,2 | 76,2 |
| Попадания в ITLB,% / ITLB hits,% | 100 | 99,9 | 100 |
| Попадания в TLB,% / TLB hits,% | 99,9 | 99,7 | 100 |
| Запросы трансляции гостевых физических адресов, млн / Guest physical address translation queries, mln | – | 69 | 0,1 |
| Гостевые обращения по физическому адресу, тыс. / Physically addressed guest memory accesses, ths | – | 119 | |
| Попадания в буфер гостевых физических адресов, % / Guest physical address buffer hits, % | – | 94,8 | 99,9 |
| Чтения ТС гипервизора из памяти, тыс. / Hypervisor PT load accesses, ths | – | 1009 | 8 |

архитектуры. Описанный подход позволил упростить систему в целом и ускорить ПС на основании информации о динамике работы программы, выработанной ФС.

Примененный при разработке подход к валидации ПС на основе сравнения с RTL и модульный принцип его построения с постепенным углублением моделирования позволили достичь относительно высоких показателей по точности (ошибка 2%) на выбранных задачах. В то же время на задачах с использованием некоторых асинхронных механизмов работы с памятью точность ПС снижается, поэтому в рамках дальнейших работ запланирована более точная их реализация.

Производительность связки ФС и ПС оказалась достаточной для запуска задач большой длительности. Были успешно внедрены механизмы моделирования с выборкой и восстановления на контрольной точке.

При первом использовании разработанного ПС были оценены два архитектурных изменения процессорного ядра «Эльбрус» шестой версии – увеличение задержки доступа в L1$ и добавление второго уровня трансляции адреса в рамках поддержки виртуализации.

В дальнейшем авторы планируют перейти к получению оценок производительности архитектуры на ПС с использованием стандартных тестовых пакетов (в частности, пакета SPEC).

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. *Akram A., Sawalha L.A.* Comparison of x86 Computer Architecture Simulators. Computer Architecture and Systems Research Laboratory (CASRL), 1, 2016 [Электронный ресурс]. URL: https://scholarworks.wmich.edu/casrl_reports/1/ (дата обращения: 02.04.2019).
2. *Порошин П.А., Мешков А.Н.,* Черных С.В. Разработка симулятора, поддерживающего потактовый режим работы, на основе текущей версии функционального симулятора архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2018. № 2. С. 69–75.
3. *Недбайло Ю.А.* Разработка сети на кристалле для перспективных многоядерных микропроцессоров // Труды МФТИ. 2017. Т. 9, № 2. С. 151–163.
4. *Недбайло Ю.А.* Проблемы масштабирования производительности подсистемы памяти многоядерного микропроцессора и методы их решения // Вопросы радиоэлектроники. 2018. № 2. С. 23–31.

5. *Знаменский Д.В.* Выбор вариантов реализации средств аппаратной поддержки виртуализации архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2014. Т. 4, № 3. С. 64–73.
6. *AMD64* Architecture Programmer's Manual Volume 2: System Programming, pp. 545–552. Advanced Micro Devices, Inc., 2018 [Электронный ресурс]. URL: https://support.amd.com/techdocs/24593.pdf (дата обращения: 24.01.2019).
7. *Intel®* 64 IA-32 Architectures Software Developer's Manual, vol. 3C, pp. 111–125. Intel Corp., 2016 [Электронный ресурс]. URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf (дата обращения: 24.01.2019).
8. *Ryckbosch F., Polfliet S., Eeckhout L.* Fast, accurate and validated full-system software simulation of x86 hardware. IEEE Micro, 2010, vol. 30, iss. 6, pp. 46–56. DOI: 10.1109/MM.2010.95.
9. *Alves M.A.Z., Villavieja C., Diener M., Moreira F.B., Navaux P.O.A.* SiNUCA: A Validated Micro-Architecture Simulator. 17th International Conference on High Performance Computing and Communications, IEEE Publ., 2015. DOI: 10.1109/HPCC–CSS-ICESS.2015.166.
10. *Ahn H. A, Li S., O S., Jouppi N.P.* McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, 2013, pp. 74–85. DOI: 10.1109/ISPASS.2013.6557148.
11. *Butko A., Garibotti R., Ost L., Sassatelli G.* Accuracy evaluation of GEM5 simulator system. 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), York, 2012, pp. 1–7. DOI: 10.1109/ReCoSoC.2012.6322869.
12. *Куцевол В.Н., Мешков А.Н., Черных С.В.* Методы оптимизации производительности программного моделирования многоядерных микропроцессоров с архитектурой «Эльбрус» // Вопросы радиоэлектроники. 2017. № 3. С. 57–61.
13. *Lafage T., Seznec A.* Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. Workload Characterization of Emerging Computer Applications, 2001, pp. 145–163.
14. *Wunderlich R.E., Wenisch T.F., Falsafi B., Hoe J.C.* SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. Proceedings of the 30th Annual International Symposium on Computer Architecture, IEEE Publ., 2003, pp. 84–97. DOI: 10.1109/ISCA.2003.1206991.
15. *Falcon A., Faraboschi P., Ortega D.* Combining Simulation and Virtualization through Dynamic Sampling. Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, IEEE Publ., 2007, pp. 72–83. DOI: 10.1109/ISPASS.2007.363738.
16. *Borgstrom G., Sembrant A., Black-Schaffer D.* Adaptive Cache Warming for Faster Simulations. Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, 2017, pp. 1–7.

## REFERENCES

1. Akram A., Sawalha L.A. Comparison of x86 Computer Architecture Simulators. Computer Architecture and Systems Research Laboratory (CASRL), 1, 2016. Available at: https://scholarworks.wmich.edu/casrl_reports/1/ (accessed 02.04.2019).
2. Poroshin P.A., Meshkov A.N., Chernyh S.V. Development of simulator with support of cycle-accurate simulation mode on base of the existing instruction set simulator of the Elbrus architecture. *Voprosy radioelektroniki*, 2018, no. 2, pp. 69–75 (In Russian).
3. Nedbailo Yu.A. On-chip network design for prospective chip multiprocessors. *Trudy MFTI*, 2017, vol. 9, no. 2, pp. 151–163 (In Russian).
4. Nedbailo Yu.A. Memory subsystem performance scaling problems in chip multiprocessors and their solution. *Voprosy radioelektroniki*, 2018, no. 2, pp. 23–31 (In Russian).
5. Znamenskiy D.V. Alternatives of hardware virtualization support inplementation for Elbrus processor architecture. *Voprosy radioelektroniki*, 2014, vol. 4, no. 3, pp. 64–73 (In Russian).
6. AMD64 Architecture Programmer's Manual Volume 2: System Programming, pp. 545–552. Advanced Micro Devices, Inc., 2018. Available at: https://support.amd.com/techdocs/24593.pdf (accessed 24.01.2019).
7. Intel® 64 IA-32 Architectures Software Developer's Manual, vol. 3C, pp. 111–125. Intel Corp., 2016. Available at: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf (accessed 24.01.2019).
8. Ryckbosch F., Polfliet S., Eeckhout L. Fast, accurate and validated full-system software simulation of x86 hardware. *IEEE Micro*, 2010, vol. 30, iss. 6, pp. 46–56. DOI: 10.1109/MM.2010.95.
9. Alves M.A.Z., Villavieja C., Diener M., Moreira F.B., Navaux P.O.A. SiNUCA: A Validated Micro-Architecture Simulator. *17th International Conference on High Performance Computing and Communications*, IEEE Publ., 2015. DOI: 10.1109/HPCC–CSS-ICESS.2015.166.
10. Ahn H. A, Li S., O S., Jouppi N.P. McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, 2013, pp. 74–85. DOI: 10.1109/ISPASS.2013.6557148.
11. Butko A., Garibotti R., Ost L., Sassatelli G. Accuracy evaluation of GEM5 simulator system. *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, York, 2012, pp. 1–7. DOI: 10.1109/ReCoSoC.2012.6322869.
12. Kutsevol V. N., Meshkov A.N., Chernykh S.V. The approaches to the performance optimization of multi-core «Elbrus» processors program models. Voprosy radioelektroniki, 2017, no. 3, pp. 57–61 (In Russian).
13. Lafage T., Seznec A. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. *Workload Characterization of Emerging Computer Applications*, 2001, pp. 145–163.

14. Wunderlich R. E., Wenisch T. F., Falsafi B., Hoe J. C. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. *Proceedings of the 30th Annual International Symposium on Computer Architecture*, IEEE Publ., 2003, pp. 84–97. DOI: 10.1109/ISCA.2003.1206991.
15. Falcon A., Faraboschi P., Ortega D. Combining Simulation and Virtualization through Dynamic Sampling. *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE Publ., 2007, pp. 72–83. DOI: 10.1109/ISPASS.2007.363738.
16. Borgstrom G., Sembrant A., Black-Schaffer D. Adaptive Cache Warming for Faster Simulations, *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, 2017, pp. 1–7.

## ИНФОРМАЦИЯ ОБ АВТОРАХ

**Знаменский Дмитрий Валерьевич**, старший инженер, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: +7 (499) 135-70-79, e-mail: znamen_d@mcst.ru.
**Куцевол Виталий Николаевич**, старший инженер-программист, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: +7 (499) 135-70-79, e-mail: kutsevol_v@mcst.ru.

## AUTHORS

**Dmitriy V. Znamenskiy**, senior engineer, MCST JSC, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-70-79, e-mail: znamen_d@mcst.ru.
**Vitaliy N. Kutsevol**, senior software engineer, MCST JSC, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-70-79, e-mail: kutsevol_v@mcst.ru.

Поступила 30.01.2019; принята к публикации 21.02.2019; опубликована онлайн 27.05.2019.
Submitted 30.01.2019; revised 21.02.2019; published online 27.05.2019.