УДК 004.414.23 DOI: 10.31114/2078-7707-2020-3-87-93

Анализ масштабируемости многоядерного микропроцессора с помощью моделирования на основе трасс событий

Ю.А. Недбайло[1,2]

[1]АО «МЦСТ», г. Москва

[2]ПАО «ИНЭУМ им. И.С. Брука», г. Москва, nonsens@inbox.ru

***Аннотация* — Развитие полупроводниковых технологий позволяет размещать всё больше процессорных ядер на одном кристалле, так что через десятилетие могут стать распространёнными процессоры с сотнями ядер. Разработчикам процессоров, соответственно, нужно добиваться хорошей масштабируемости их дизайна для наращивания числа ядер. Для этого требуются конструктивные решения и оптимизации, повышающие масштабируемость, а также набор инструментов, позволяющий оценивать их влияние на производительность. В данной статье описывается методика моделирования, разработанная для процессоров семейства Эльбрус, и оценки производительности таких процессоров, полученные на 64- и 256-ядерных моделях.**

***Ключевые слова* — архитектура «Эльбрус», программное моделирование, потактовый симулятор, многоядерность, подсистема памяти, общий кэш.**

I. Введение

Развитие полупроводниковых технологий позволяет разрабатывать микропроцессоры со всё большим количеством ядер. При этом, в связи с растущей сложностью схем, требуется их высокая переиспользуемость в последующих поколениях процессоров, то есть масштабируемость. Для оценки масштабируемости схем требуется соответствующий инструментарий, в частности методы и средства оценки их влияния на производительность в реальных задачах.

Некоторые схемы и параметры известны из литературы и накопленного опыта, например то, что стоит реализовать распределённый общий кэш, соединённый с ядрами сетью на кристалле [1] (рис. 1). Некоторые требуют моделирования, например, оптимальная пропускная способность памяти зависит от частоты обращений ядер в память, частоты промахов в кэши и других характеристик конкретной задачи.

Поскольку наращивание числа ядер в основном подразумевает модификацию подсистемы памяти процессора, главным образом общего кэша и сети на кристалле, то модель должна достаточно точно отражать влияние их деталей на производительность.

II. Существующие решения

Точные средства моделирования микропроцессоров – программное моделирование RTL и прототипирование на ПЛИС – не подходят для оценки производительности реальных задач, поскольку используют полное описание процессора на языке описания аппаратуры и потому либо слишком медленны, либо требуют крайне дорогостоящего оборудования.

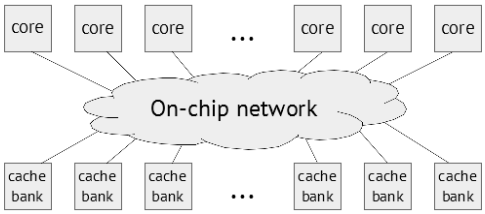


Рис. 1. Идея микропроцессора с распределённым общим кэшэм

Ближайшей альтернативой являются упрощённые потактовые симуляторы. Например, Gem5 [2] поддерживает различные наборы команд, имеет очень гибкую модель подсистемы памяти и разные уровни точности. Самые точные модели широко используются в исследованиях для моделирования порядка десятков ядер, но далее их ограничивает скорость и требуемый объём памяти. Для повышения скорости был разработан ряд более упрощённых симуляторов, например, ZSim [3] способен моделировать 1024-ядерный процессор со скоростями до 1500 MIPS на обычном 16-ядерном процессоре, но эти скорости достаются ценой таких упрощений как ядер, так и подсистемы памяти, которые ставят под вопрос их пригодность для обсуждаемой задачи.

Для детального моделирования подсистемы памяти процессоров с большим числом ядер применяются симуляторы на основе трасс событий. Их идея состоит в разделении моделирования на две фазы. В первой программа выполняется на точной модели ядер с упрощённой моделью подсистемы памяти и в файл-трассу записываются все события определённого типа; это могут быть исполняемые команды [4], принятые сетью-на-кристалле пакеты [5] или обращения программы в память. На второй фазе соответствующая событиям часть подсистемы памяти процессора моделируется воспроизведением этих событий из трассы. Таким образом для анализа подсистемы памяти не нужно тратить ресурсы на многократное моделирование выполнения программы ядрами –

трассу каждого теста достаточно снять один раз и затем использовать для моделирования всех интересующих конфигураций.

Трасса событий должна отображать зависимости между ними, иначе не будет моделироваться влияние задержек. Для трасс команд, применяемых для моделирования out-of-order процессоров [4], и трасс пакетов, подходящих для моделирования сетей-на-кристалле [5], соответствующие методы известны. Для моделирования процессоров «Эльбрус» больше подойдут трассы обращений в память. Мы разработали методику снятия таких трасс и соответствующий симулятор подсистемы памяти процессора.

III. Методика снятия трасс событий

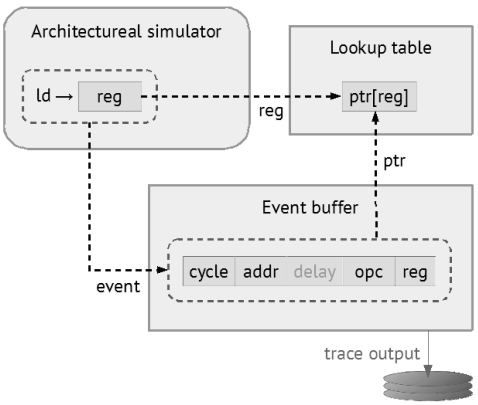
Для снятия трасс обращений в память с зависимостями между ними был доработан функциональный симулятор микропроцессора «Эльбрус» [6]. Этот симулятор пока не поддерживает многопоточные программы в пользовательском режиме, поэтому далее в статье подразумеваются однопоточные задачи. Как только поддержка появится, описываемая методика и симулятор подсистемы памяти могут быть соответственно дополнены такой методикой, как SynchroTrace [7].

Команды в качестве событий подходят при моделировании процессоров с внеочередным исполнением инструкций [4]. «Эльбрус» – это VLIW-процессоры и на сегодняшний день они не используют внеочередное исполнение, поэтому такая точность трасс в их случае не требуется. Другим распространённым вариантом является использование пакетов сети на кристалле в качестве событий [5], но он не подходит для анализа влияния параметров кэшей на производительность. Использование обращений в память в качестве событий позволяет в случае «Эльбруса» моделировать все интересующие устройства с достаточной точностью.

Доработка функционального симулятора для снятия трасс включает буфер событий, хранящий время, адрес, код операции, время до первой зависимой команды и регистр назначения, а также таблицу соответствия регистров позициям в этом буфере (рис. 2). При обращении программы в память, в буфере создаётся событие и в таблицу добавляется указатель на него; при использовании регистра, в таблице проверяется наличие указателя, находится соответствующее обращение в память, и в него записывается информация о зависимости в виде разницы во времени.

Снятая полностью, такая трасса может получиться большой, в несколько терабайт для одного теста даже в сжатом виде. Для сокращения размера трассы реализовано её снятие случайными интервалами (то есть, кластерной выборкой [8]), затем интервалы анализируются и выбираются один или несколько самых репрезентативных. Оптимальный размер интервала определяется временем, требуемым моделируемой системе на «разогрев», и желаемой точностью. В рассматриваемых микропроцессорах, на одно ядро приходится около 2 МБ кэшей, что по пессимистичным оценкам требует порядка 3 млн. тактов. Для точности порядка 1%, соответственно, выбран интервал в 300 млн. тактов. Количество интервалов определяет степень их соответствия программе: из литературы известно, что для точности в несколько процентов при случайной выборке требуется порядка ста [8], а после анализа можно ограничиться выбором от двух до десяти [9].

а)



б)

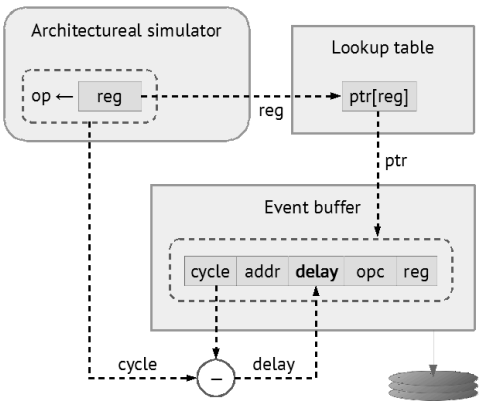


Рис. 2. Снятие трассы: а) обращение в память, б) использование регистра

Скорость работы симулятора в несколько мегагерц позволяет снять трассы всего пакета SPEC CPU2006 на обычном 16-ядерном процессоре примерно за неделю.

Описанная методика может быть реализована и в других in-order архитектурах.

IV. Модель многоядерного процессора

Разработанный для детального моделирования подсистемы памяти симулятор многоядерного процессора включает упрощённые модели ядер, воспроизводящие трассы, и детальные модели кэшей, сети соединений на кристалле и контроллера памяти. Результатом его работы является различная статистика (рис. 3).

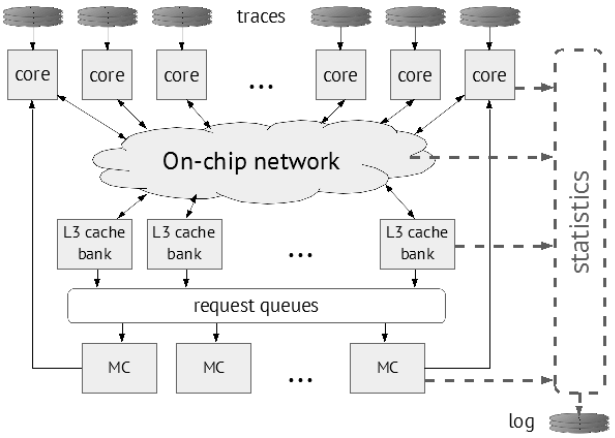


Рис. 3. Структура симулятора микропроцессора

Модели ядер воспроизводят обращения программы в память из соответствующих файлов трасс и моделируют блокировки конвейера, вызываемые зависимости между этими обращениями. Модели кэшей (L1i, L1d, L2, L3) имеют настраиваемый размер, ассоциативность, инклюзивность, алгоритмы замещения, разбиения и миграции [10]. Их задержки моделируются в первом приближении самим симулятором, поэтому достаточно простейшей поведенческой реализации, что позволяет достичь высокой скорости моделирования.

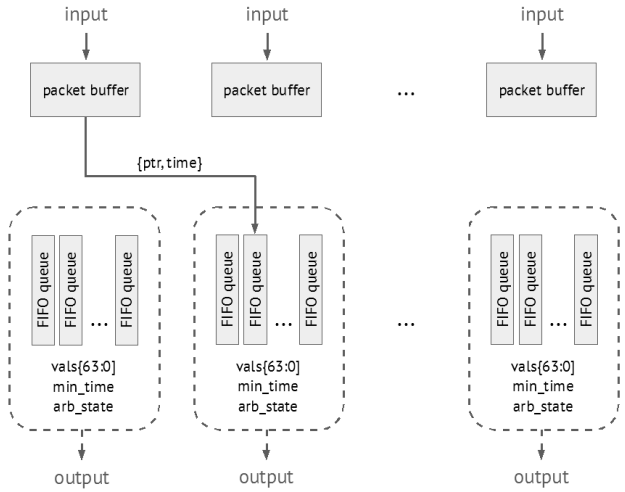


Рис. 4. Структура модели сети соединений на кристалле

Модель сети соединений также предельно упрощена, но при этом способна отражать пропускную способность, задержки и честность реальной сети, что и требуется для оценки производительности микропроцессора в целом. На каждом входе модели сети – очередь пакетов, на каждом выходе 64 очереди указателей на них (для разных расстояний между входом и выходом) и три переменные, использование которых позволяет тратить минимум времени на проверку очередей (рис. 4). Арбитры на выходах, счётчики пакетов и механизм типа source throtling [11] позволяют моделировать требуемую честность сети и зависимость задержек от интенсивности трафика без заметного ущерба для скорости моделирования. Для контроля точности модели ведётся статистика загруженности каналов сети при предполагаемой топологии сети (сетка) и детерминированной (XY) маршрутизации.

Модель контроллера памяти также упрощена, поскольку в многоядерных процессорах поток обращений в память выглядит всё более случайным с увеличением количества ядер [12]; для повышения точности, например, для исследования оптимизаций, уменьшающих эту случайность, в симулятор планируется добавить точный симулятор памяти Ramulator [13].

Данный симулятор также подойдёт для моделирования других in-order архитектур. Точность и скорость его работы исследованы в [14]. На 16-ядерном Xeon E5 2698 v3 моделирование всех 29 тестов пакета SPEC CPU2006 в 1024-ядерной конфигурации занимает примерно три дня; конфигурации с меньшим числом ядер моделируются пропорционально быстрее.

V. Результаты экспериментов

С помощью описанной методики была оценена производительность 64- и 256-ядерных процессоров в конфигурациях, аналогичных использованным в [14] для 1024-ядерных тестов. В данной работе мы более подробно изучим влияние деталей реализации общего кэша и оценим масштабируемость производительности. Память предполагается 8-канальной с пропускной способностью в 16 и 32 байта за такт ядра на канал, соответственно. На диаграммах показаны результаты десяти тестов с наибольшей разницей в результатах. Производительность даётся относительно однопоточной в 8-ядерной конфигурации или относительно аналогичной конфигурации без разбиения кэша.

A. Инклюзивность

В [15] на основании 16-ядерных тестов сделан вывод, что стоимость реализации отдельного справочника, необходимого для оптимизаций времени доступа, при любом количестве ядер оправдана повышением эффективности кэша за счёт отказа от инклюзивности. Мы проверили это при 64 и 256 ядрах и получили среднее преимущество неинклюзивной схемы в 3.5 и 4 процента, соответственно (рис. 5). Учитывая возможность дальнейшего почти бесплатного повышения эффективности оптимизациями типа FLEXclusion, цена справочника по-прежнему выглядит оправданной.

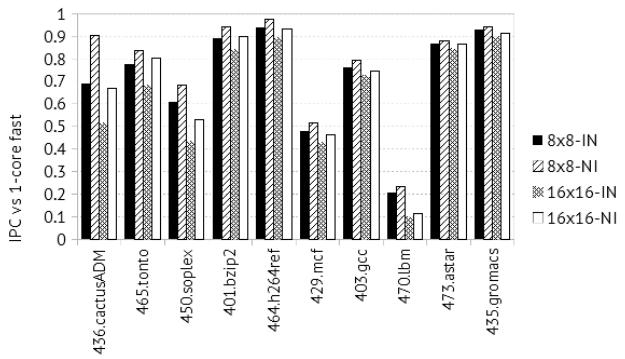


Рис. 5. Производительность 64- и 256-ядерных моделей при разной инклюзивности L3 кэша

Результаты этих тестов, нагружающих каждое ядро, также показывают падение средней производительности ядра на 23% при 64 ядрах и ещё на 13% при увеличении числа ядер до 256, что означает прирост суммарной производительности ядер в 49 раз при переходе от одного ядра к 64 и в 3.48 раза от 64 к 256. Это позволяет сделать вывод о хорошем масштабировании производительности в хорошо распараллеливаемых задачах вплоть до 256 ядер.

B. Ассоциативность и разбиение

В [16] мы показали преимущество ZCache с механизмом разбиения Futility Scaling над традиционным way-partitioning. Однако в [14] простой механизм Partitioning First с традиционной ассоциативностью продемонстрировал производительность почти идентичную ZCache. Проверим гипотезу, что при инклюзивной схеме разница должна быть больше (рис. 6).

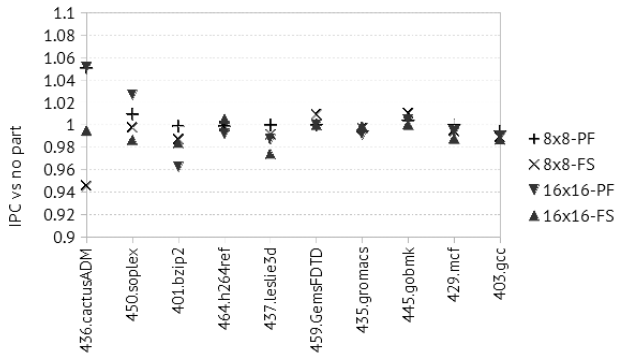


Рис. 6. Производительность 64- и 256-ядерных моделей при разном разбиении L3 кэша

Действительно, разница стала более заметной, но по-прежнему не превышает нескольких процентов. Напрашивается вывод, что в рассматриваемых конфигурациях ассоциативность общего кэша не так важна, как в конфигурациях из публикаций про ZCache [17]. Мы проверили на 64-ядерной модели, как на результаты влияет количество столбцов в кэше (рис. 7).

При восьми и менее столбцах, Partitioning First демонстрирует ожидаемое падение

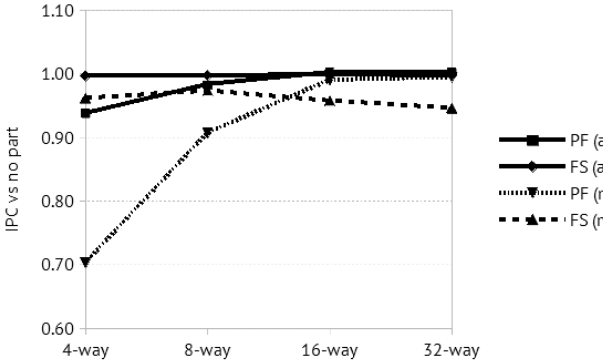


Рис. 7. Производительность 64-ядерной модели при разных ширине и разбиении L3 кэша

производительности, а при 16 и более столбцах разницы почти нет. Возможно, в более сложных сценариях она проявится, а пока механизм Partitioning First выглядит оптимальным выбором за счёт своей простоты и чуть лучшей производительности при 16 и более столбцах.

В любом случае, реализация механизма разбиения и, соответственно, любых политик качества обслуживания (QoS) на его основе выглядит возможной без заметного ущерба для производительности и больших затрат оборудования при любом количестве ядер.

C. Уменьшение времени доступа

В [14] мы обнаружили двухкратное падение производительности 1024-ядерной модели относительно 8-ядерной в некоторых сценариях из-за увеличения времени доступа в память. Это ставит под вопрос целесообразность такого наращивания числа ядер. Повторим эти эксперименты на 64- и 256-ядерных моделях и проверим, насколько может помочь двукратное уменьшение задержек сети (реализацией экспресс-виртуальных каналов или/и выбором другой топологии) (рис. 8).

На 64-ядерной модели с общим кэшем без уменьшения задержек сети (Shared и Hybrid) однопоточные тесты демонстрируют в среднем на 1–2% большую скорость, чем на модели 8-ядерного процессора, с максимальным отставанием в 14%. Уменьшение задержек (Hybrid-N1) даёт среднее преимущество в 8% и максимальное отставание в 2%. На основании этих результатов можно сделать вывод, что со всеми оптимизациями 64-ядерный процессор будет почти однозначно быстрее 8-ядерного даже в однопоточных задачах.

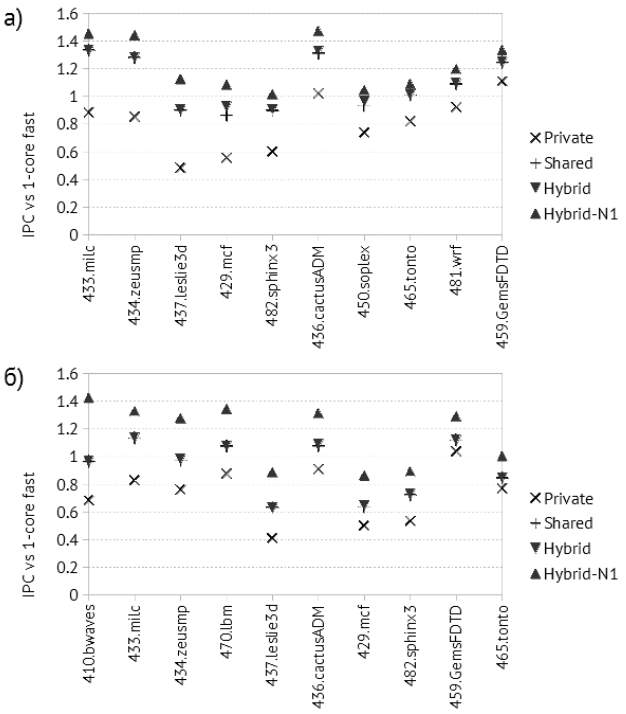


Рис. 8. Производительность в однопоточных сценариях а) 64-ядерной модели б) 256-ядерной

На 256-ядерной модели с общим кэшем падение производительности достигает 37% при среднем в 6,5%. Уменьшение задержек сети исправляет ситуацию: максимальное отставание от 8-ядерной модели уменьшается до 14%, а в среднем наблюдается прирост в 4,6%. В целом, 256-ядерность можно считать оправданной, а учитывая, что при разработке новых процессоров часто улучшают микроархитектуру или повышают частоту ядер, эти результаты позволяют ожидать, что и 256-ядерный процессор будет однозначно быстрее 8-ядерного при должной оптимизации.

VI. Заключение

Для оценки производительности многоядерных процессоров с сотнями ядер можно использовать методику, включающую снятие трасс событий доработанным потактовым симулятором процессора с упрощённой моделью подсистемы памяти, и затем воспроизведение этих трасс в детальной модели подсистемы памяти исследуемого многоядреного процессора.

Эксперименты на 64- и 256-ядерных моделях процессоров в многопоточных сценариях показывают средний прирост производительности в 49 раз у 64 ядер относительно одного ядра 8-ядерной модели и ещё в 3.48 раза при увеличении до 256, что позволяет сделать вывод о хорошем масштабировании производительности в хорошо распараллеливаемых задачах. В однопоточных тестах на тех же моделях падение производительности из-за увеличения времени доступа в память, компенсируемое увеличением объёма общего кэша и оптимизациями времени доступа, составляет не более 2% и 14%, соответственно; в среднем производительность повышается на 8% и 4,6%.

На основании результатов этих экспериментов, имитирующих работу процессора в наиболее и наименее распараллеливаемых задачах, можно сделать вывод о возможности наращивания числа ядер в процессорах семейства «Эльбрус» вплоть до 256-ти с адекватным увеличением производительности в целом и без значительного её падения почти во всех задачах. То есть, вопреки распространённому мнению, разработка универсальных процессоров с сотнями ядер на обычных кремниевых технологиях имеет смысл.

Эти результаты, однако, получены на упрощённой модели и ограниченном наборе тестов. Для определения оптимальной конфигурации таких процессоров следует повторить эксперименты на более точной модели и в более актуальных и разнообразных задачах, когда технологии позволяющие достичь такого количества ядер станут доступны и будут известны спецификации соответствующих типов памяти.

Литература

[1] Yu. A. Nedbailo. Avoiding common scalability pitfalls in shared-cache chip multiprocessor design // In 2019 International Conference on Engineering and Telecommunication (EnT), Nov 2019.

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, TusharKrishna, Somayeh Sardashti, and et al. // The Gem5 simulator. SIGARCH Comput. Archit. News, 39(2):17, August 2011.

[3] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems // In Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA 13, page 475–486, New York, NY, USA, 2013. Association for Computing Machinery.

[4] R. Jagtap, S. Diestelhorst, A. Hansson, M. Jung, and N. When. Exploring system performance using elastic traces: Fast, accurate and portable // In 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), pages 96–105, July 2016.

[5] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace: Dependency-driven trace-based network-on-chip simulation // In Proceedings of the Third International Workshop on Network on Chip Architectures, NoCArc '10, pages 31–36, New York, NY, USA, 2010. ACM.

[6] 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.

[7] Karthik Sangaiah, Michael Lui, Radhika Jagtap, Stephan Diestelhorst, Siddharth Nilakantan, Ankit More, Baris Taskin, and Mark Hempstead SynchroTrace: Synchronization-aware architecture-agnostic traces for lightweight multicore simulation of CMP and HPC workloads // ACM Trans. Archit. Code Optim., 15(1), March 2018.

[8] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors // In Proceedings International Conference on

Computer Design. VLSI in Computers and Processors, pages 468–477, Oct 1996.
[9] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior // In Proceedingsof the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS X, page 45–57, New York, NY, USA, 2002. Association for Computing Machinery.
[10] Кожин А. С., Недбайло Ю. А. Методы оптимизации времени доступа в общий кэш многоядерного микропроцессора // Вопросы радиоэлектроники. 2017. No 3. С. 27–32.
[11] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt. Fairness via source throttling: A configurable and high-performance fairness substrate for multicore memory systems // ACM Trans. Comput. Syst., 30(2):7:1–7:35, April 2012.
[12] Yu.A. Nedbailo and Igor A. Petrov. Increasing DDR4 SDRAM throughput in parallel workloads // In 2020 Moscow Workshop on Electronic and Networking Technologies (MWENT), Mar 2020.
[13] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible DRAM simulator // IEEE Computer Architecture Letters, 15(1):45–49, Jan 2016.
[14] Y. Nedbailo. Fast and scalable simulation framework for large in-order chip multiprocessors // In 2020 26th Conference of Open Innovations Association (FRUCT), Yaroslavl, Russia, April 2020, pp. 335–345.
[15] Кожин А.С., Недбайло Ю.А. Оптимизация общего кэша третьего уровня микропроцессора «Эльбрус-8С» // Вопросы радиоэлектроники. 2015. № 3(3). С. 21–30.
[16] Недбайло Ю.А. Проблемы масштабирования производительности подсистемы памяти многоядерного микропроцессора и методы их решения // Вопросы радиоэлектроники. 2018. № 2. С. 23–31.
[17] Ruisheng Wang and Lizhong Chen. Futility scaling: High-associativity cache partitioning // In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47, pages 356–367, Washington, DC, USA, 2014. IEEE Computer Society.

Multiprocessor Chip Scalability Analysis Using Trace-Based Simulation

Yu.A. Nedbailo[1,2]

[1]MCST JSC, Moscow

[2]INEUM im.I.S.Bruka, nonsens@inbox.ru

***Abstract* — As chip technology advances, the number of cores in mainstream chip multiprocessors (CMP) increases, so chips with hundreds of cores may become common within a decade. One of the challenges this trend sets to computer architects is to make the current CMP designs scalable to larger numbers of cores. A set of optimizations and design decisions needs to be developed accordingly and a tool set that would allow us to predict how they may affect the performance of larger CMPs is therefore necessary. In this paper, we describe a trace-based simulation framework we devised for Elbrus microprocessor family and the estimates of how this microarchitecture may perform in 64- and 256-core configurations.**

***Keywords* — Elbrus processor architecture, software simulation, cycle-accurate simulator (CAS), many-core, memory subsystem, shared cache.**

References

[1] Yu. A. Nedbailo. Avoiding common scalability pitfalls in shared-cache chip multiprocessor design // In 2019 International Conference on Engineering and Telecommunication (EnT), Nov 2019.
[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, TusharKrishna, Somayeh Sardashti, and et al. // The Gem5 simulator. SIGARCH Comput. Archit. News, 39(2):17, August 2011.
[3] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems // In Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA 13, page 475–486, New York, NY, USA, 2013. Association for Computing Machinery.
[4] R. Jagtap, S. Diestelhorst, A. Hansson, M. Jung, and N. When. Exploring system performance using elastic traces: Fast, accurate and portable // In 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), pages 96–105, July 2016.
[5] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace: Dependency-driven trace-based network-on-chip simulation // In Proceedings of the Third International Workshop on Network on Chip Architectures, NoCArc '10, pages 31–36, New York, NY, USA, 2010. ACM.
[6] P.A. Poroshin, A.N. Meshkov, and S.V. Chernyh. 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. № 2. pp. 69–75 (in Russian).
[7] Karthik Sangaiah, Michael Lui, Radhika Jagtap, Stephan Diestelhorst, Siddharth Nilakantan, Ankit More, Baris Taskin, and Mark Hempstead. SynchroTrace: Synchronization-aware architecture-agnostic traces for lightweight multicore simulation of CMP and HPC workloads // ACM Trans. Archit. Code Optim., 15(1), March 2018.
[8] T.M. Conte, M.A. Hirsch, and K.N. Menezes. Reducing state loss for effective trace sampling of superscalar processors // In Proceedings International Conference on Computer Design. VLSI in Computers and Processors, pages 468–477, Oct 1996.
[9] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior // In Proceedingsof the 10th International

Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS X, page 45–57, New York, NY, USA, 2002. Association for Computing Machinery.

[10] A.S. Kozhin and Yu.A. Nedbailo. Methods of shared cache access latency optimization in chip multiprocessors // Voprosy radioelektroniki. 2017. № 3, pp. 27–32 (in Russian).

[11] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt. Fairness via source throttling: A configurable and high-performance fairness substrate for multicore memory systems // ACM Trans. Comput. Syst., 30(2):7:1–7:35, April 2012.

[12] Yu.A. Nedbailo and Igor A. Petrov. Increasing DDR4 SDRAM throughput in parallel workloads // In 2020 Moscow Workshop on Electronic and Networking Technologies (MWENT), Mar 2020.

[13] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible DRAM simulator // IEEE Computer Architecture Letters, 15(1):45–49, Jan 2016.

[14] Y. Nedbailo. Fast and scalable simulation framework for large in-order chip multiprocessors // In 2020 26th Conference of Open Innovations Association (FRUCT), Yaroslavl, Russia, April 2020, pp. 335–345.

[15] A. Kozhin, Y. Nedbailo. Optimizing the inclusive shared L3 cache in «Elbrus-8C» microprocessor // Voprosy radioelektroniki. 2015. № 3(3), pp. 21–30 (in Russian).

[16] Nedbailo Yu. A. Memory subsystem performance scaling problems in chip multiprocessors and their solution // Voprosy radioelektroniki. 2018. № 2, pp. 23–31 (in Russian).

[17] Ruisheng Wang and Lizhong Chen. Futility scaling: High-associativity cache partitioning // In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47, pages 356–367, Washington, DC, USA, 2014. IEEE Computer Society.