ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
СПбГУТ))) Сентябрь 2016 Том 4 № 3 ISSN 2307-1303 http://www.itt.sut.ru
TELECOM IT
September 2016 Vol. 4 Iss. 3 ISSN 2307-1303 http://www.itt.sut.ru

# МЕТОДЫ ОЦЕНКИ ТРАФИКА В СОВРЕМЕННЫХ МНОГОЯДЕРНЫХ ПРОЦЕССОРАХ ОБЩЕГО НАЗНАЧЕНИЯ

К. Д. Борунова[1]

[1] СПбГУТ, Санкт-Петербург, 193232, Российская Федерация
* Адрес для переписки: karolinadm@mail.ru

**Аннотация**
**Предмет исследования.** В статье рассмотрены методы оценки трафика в современных многоядерных процессорах общего назначения. **Метод.** В качестве метода исследования приводится анализ существующих подходов для изучения трафика на интерфейсах Центрального Процессора (ЦП). **Основные результаты.** В статье приведены результаты исследования трафика, которые были получены описанными методами оценки. **Практическая значимость.** Используя всю информацию, полученную приведенными методами, можно вывести детальную модель трафика современных ЦП, которая должна быть принята во внимание при разработке новых методов внутренней коммуникации ЦП.

**Ключевые слова**
многоядерные системы, центральный процессор, кэш-память, анализ трафика, стохастические процессы.

**Информация о статье**
УДК 004.2
Язык статьи – русский.
Поступила в редакцию 01.07.16, принята к печати 26.08.16.

**Ссылка для цитирования:** Борунова К. Д. Методы оценки трафика в современных многоядерных процессорах общего назначения // Информационные технологии и телекоммуникации. 2016. Том 4. № 3. С. 31–39.

# METHODS FOR TRAFFIC EVALUATION ON MODERN MULTI-CORE PROCESSORS

K. Borunova[1*]

[1] SPbSUT, St. Petersburg, 193232, Russian Federation
* Corresponding author: karolinadm@mail.ru

**Abstract—Research subject.** The article considers the methods for traffic evaluation on modern multi-core processors. **Method.** As a method of research provides an analysis of existing approaches to examine traffic on interfaces of the central processing unit (CPU). **Core results.** The article presents the results of a study of traffic that were obtained as described by methods of assessment. **Practical relevance.** Using all information received as described methods, you can display a detailed traffic model of modern CPUS, which should be taken into account in the development of new methods of internal communication CPU.
**Keywords—**Multicore CPUs, modern CP-CPUs, cache-memory, traffic analysis, stochastic processes.

**Article info**
Article in Russian.
Received 01.07.16, accepted 26.08.16.

**For citation:** Borunova K.: Methods for Traffic Evaluation on Modern Multi-Core Processors // Telecom IT. 2016. Vol. 4. Iss. 3. pp. 31–39 (in Russian).

## Введение

Производство центральных процессоров (ЦП) достигло уровня, когда стало выгоднее увеличивать производительность «горизонтально», т. е. распараллеливать вычисления, а не продолжать увеличивать тактовую частоту. Основные производители ЦП, такие как Intel и AMD, впервые представили в 2005 г. двухъядерный процессор. Начав с простого внедрения двух вычислительных узлов (ядер) на одном процессоре и обеспечения общего доступа к запоминающему устройству (ОЗУ), сегодня Intel и AMD выпускают процессоры с 4, 8 и более ядрами на одном кристалле с глубокой интеграцией компонентов и динамических потоков, распределённых между ядрами[1,2]. Однако, при увеличении количества ядер необходимо обеспечивать эффективное взаимодействие всех протекающих процессов, которое осуществляется через специализированные области общей памяти (т. е. обращение идет к основной памяти и кэшу). При разработке новых методов внутренней коммуникации ЦП необходимо определить модель трафика.

Основной целью данной статья является обзор методов оценки трафика в современных многоядерных процессорах общего назначения для последующего определения модели трафика. Данная статья имеет следующую структуру: во втором разделе представлено краткое описание архитектуры x86 и иерархической структуры кэш-памяти в процессорах общего назначения, третий раздел посвящен описанию методологии оценки трафика, далее представлены результаты исследования, в пятом разделе перечислены основные выводы.

## Архитектура процессоров x86 и структура кэш-памяти

На базе процессоров архитектуры x86 уже более 30 лет создаются вычислительные системы различной сложности – от персональных компьютеров и устройств до мощных серверов, вычислительных кластеров и суперкомпьютеров. В настоящее время данная архитектура, представленная на рис. 1, а также

[1] Intel Core i7-5960x processor extreme edition (Technical specifications). URL: http://ark.intel.com/products/82930/Intel-Core-i7-5960XProcessor-Extreme-Edition-20M-Cache-up-to-3 50-GHz
[2] AMD FX Series Processors (Technical specifications). URL: http://www.amd.com/en-us/products/processors/desktop/fx

ее варианты являются одними из доминирующих на рынке микропроцессоров общего назначения.

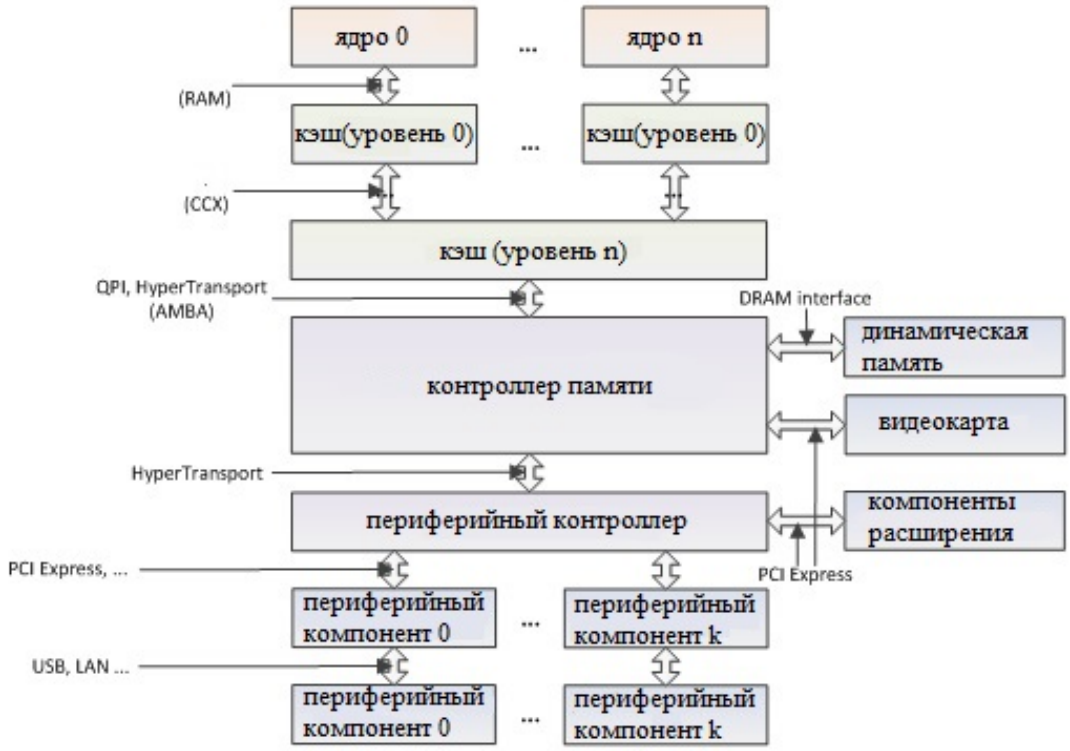


Рис. 1. Структура современной вычислительной системы на базе архитектуры x86

В системе используются следующие компоненты:

1. Ядро вычислительной системы, включающее в себя:
   а) Процессоры, поддерживающие один или более поток управления.
   б) Кэш-память нескольких уровней (обычно до 3-х).
   в) Контроллер памяти, отвечающий за маршрутизацию запросов к памяти.
   г) Основная память.
2. Концентратор ввода/вывода, служащий для подключения дополнительных компонентов к ядру вычислительной системы.
3. Компоненты, расширяющие возможности вычислительной системы.
4. Периферийные контроллеры.

Кэш-память организована по иерархическому принципу, особенностью которого является использование нескольких уровней памяти, различных по объёму и времени отклика. Кэш предназначен для непродолжительного хранения информации и представляет собой высокоскоростную память, состоящую из блоков определённого размера, которые называются «кэш-линия». Представленная на рис. 2 многоуровневая модель организации кэш-памяти включает в себя, как минимум, три уровня кэша [1, 2].

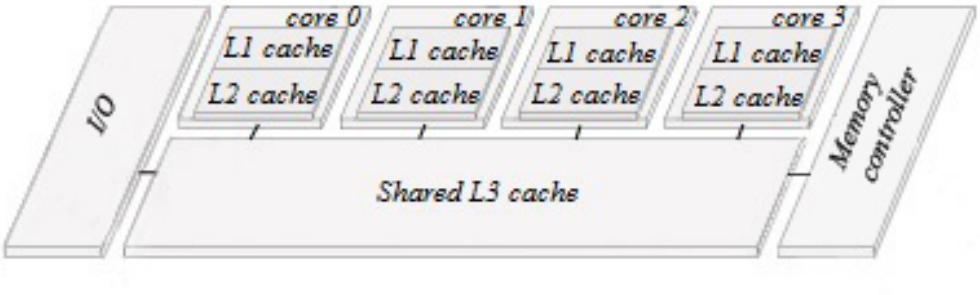


Рис. 2. Многоуровневая модель организации кэш-памяти

Каждое ядро имеет собственный кэш уровней *L*1 и *L*2, а кэш уровня *L*3 уже является общим для всех ядер. Время отклика на каждом уровне кэш-памяти различно, но не превышает 10 нс. Когда ядра отсылают запрос на чтение/запись данных к запоминающему устройству, кэш-контроллер использует т. н. алгоритм предсказания времени хранения данных, которые могут быть использованы в дальнейшем. Наиболее запрашиваемые данные хранятся в кэш-памяти уровней *L*1, *L*2. Существует также протокол когерентности, который следит за тем, чтобы операции с одним и тем же элементом данных выполнялись на разных процессорах последовательно, а устаревшие копии удалялись.

## Методология оценки трафика

Существуют три различных подхода для изучения трафика на интерфейсах ЦП, таких как *L*1 – *L*2, *L*2 – *L*3, *L*3 – DRAM и в обратном направлении:

1) аналитический метод протоколов когерентности кэша;
2) косвенные измерения;
3) симуляция системы.

Используя первый метод, можно попытаться определить характеристики трафика, анализируя логику различных компонентов ЦП и подсистемы кэш-памяти. Данная информация может быть представлена в документах, размещённых, например, на сайтах компаний-производителей ЦП. Несмотря на то, что не всегда хватает некоторых точностей и деталей, такой анализ может стать первым шагом в исследовании. Но существуют, как минимум, две проблемы, которые могут помешать в получении детального анализа трафика. Во-первых, информация об особенностях архитектуры не часто подробно представлена. Например, пропускная способность внутренних компонентов ядра зависит от динамики вычислительного процесса, а в официальной документации представлены результаты т. н. «идеального» поведения. Так же теоретическое знание работы протоколов когерентности кэш-памяти не всегда является определением того, каким образом контроллеры взаимодействуют при выполнении транзакций. Во-вторых, статическая оценка не предоставляет информации о динамике трафика, т. е. о распределении нагрузки по времени. Поэтому, помимо логического анализа, необходимо проводить косвенные измерения и/или симулировать определённую систему или её части.

Второй метод основан на косвенных измерениях характеристик трафика. В современных ЦП не существует непосредственного механизма для измерения трафика на определённых интерфейсах. Однако, используя ЦП компании Intel, можно рассчитать объём трафика, используя т. н. «счётчики производительно-

сти». Данные счётчики предоставляют информацию о количестве пропущенных адресов на каждом уровне кэш-памяти в заданном интервале времени, по которым можно судить о загруженности процессора. Для такого расчёта можно использовать приложения perf[3] and Intel Performance Counter Monitor[4] (PCM), которые находятся в открытом доступе. Т. к. информация о синхронизации недоступна, получение детальной структуры трафика (временные ряды событий) на внутренних интерфейсах ЦП невозможно.

Ниже приведён пример получения объём трафика на интерфейсах *L2–L3* с помощью программы PCM. График, построенный после получения численных значений количества передаваемых данных для теста AES, представлен на рис. 3.

Общий интерфейс *L3* используется только в случае, если ни одной записи (линии) не было найдено на уровнях *L*1, *L2* кэш-памяти. В этом случае всегда выполняется запрос на интерфейс *L3*. Обрабатывая этот запрос, кэш-контроллер *L3* находит запрошенные данные в кэш-памяти *L3* или запоминающем устройстве и отправляет их обратно на кэш-уровень *L2*. Общее количество записей за единицу времени соответствует количеству пропусков на *L2* за текущий период. В 64-битной архитектуре ЦП размер кэш-линии составляет 64 байта, а длина запроса на чтение – 8 байт. Таким образом, трафик на общем интерфейсе *L3* вычисляется как $T = L2_m(8 + 64)$, где $L2_m$ – количество пропущенных данных за единицу времени на *L2*.

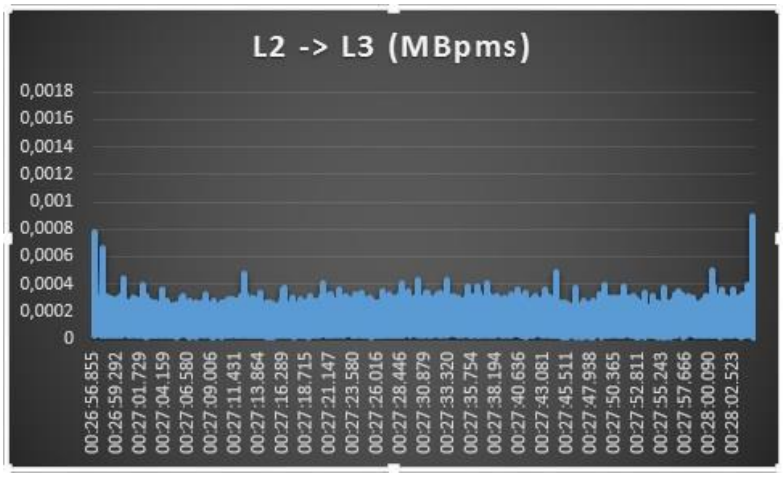


Рис. 3. Трафик в определённый момент времени

Последний метод позволяет детализировать структуру трафика на внутренних интерфейсах с помощью симуляции ЦП, используя, например, симуляторы, поддерживающие архитектуру x86, MARSSx86[5], Gem5 [3], zSim [4], SST[6]. Для определения характеристик трафика была создана типичная x86 архитектура ЦП с соответствующей подсистемой кэш-памяти в симуляторе Gem5.

[3] Perf: Linux Profiling with Performance Counters. URL: https://perf.wiki.kernel.org
[4] Intel Performance Counter Monitor. URL: http://www.intel.com/software/pcm
[5] MARSSx86 – Micro-Architectural and System Simulator for x86-based systems. URL: http://marss86.org/ marss86/
[6] The structural simulation kit (SST). URL: http://www.ece.cmu.edu/calcm/isca2015

Для эмуляции типичной нагрузки на внутренних интерфейсах ЦП были выбраны тесты, которые затрагивают различные аспекты при выполнении программного кода: чтение, запись, сортировка, комплексный рекурсивный расчёт факториала, алгоритм нахождения наибольшего общего делителя (алгоритм Евклида), алгоритм шифрования/дешифрования AES и сжатие данных zlib. Количество одновременно выполняющихся тестов соответствует числу ядер. В целом, было произведено 70 тестов. Результаты симуляции сохранены в формате *.vcd, для получения данных временных рядов выбранные объекты были сохранены в формате *.*tim* и преобразованы с помощью написанной программы на языке C. Все тесты выполнены за 20 минут с результатом в 6 000 примеров. Для всех тестов было произведено два запуска. Все ядра выполняли вычисления одновременно, используя специфику тестов.

Для проведения анализа трафика был использован 8ми-ядерный процессор Intel Core i7-5960X архитектуры Haswell с 20 МБ кэш-памяти *L*3. Тактовая частота была выбрана равной 3.0 ГГц. Для доступа к счётчикам производительности использовался Intel Performance Counter Monitor (PCM). Так как вычисления производились под работающей операционной системой (ОС), полученные данные содержат некоторое количество информации, генерируемой подпрограммами ОС. К тому же PCM осуществляет дополнительную нагрузку на ЦП.

## Результаты исследования

На рис. 4 представлены пример измеренного трафика на интерфейсах *L*2–*L*3 и *L*3–DRAM при выполнении алгоритма шифрования/дешифрования *AES* для разного количества ядер с использованием индикаторов «занятости интерфейса» $IA + b$, где $A$ – событие «занятый интерфейс», $b$ – константа, обозначающая количество ядер.

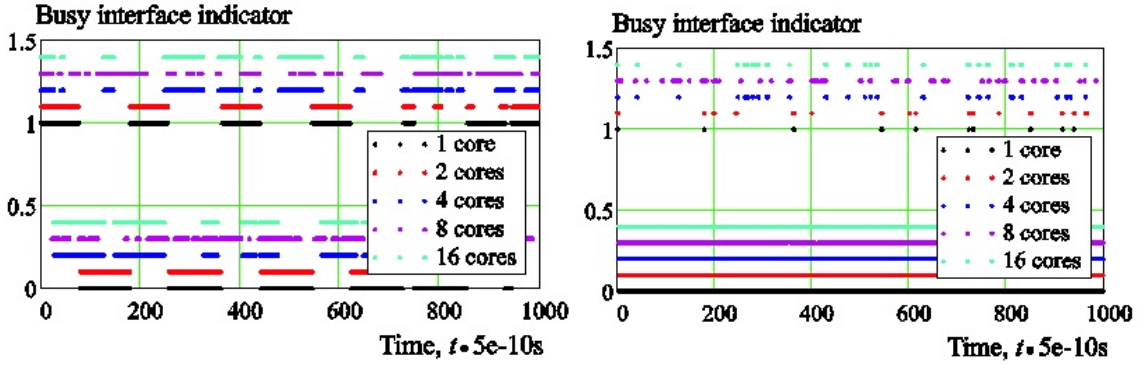


Рис. 4. Трафик на интерфейсе *L*2–*L*3 (слева), *L*3–DRAM (справа)

На основе проведённого анализа можно делать два вывода: (1) трафик на обоих интерфейсах имеет стохастическую структуру и (2) трафик чётко разделён на группы (серии).

Приведённые выше выводы имеют ряд важнейших следствий для будущих исследований ЦП. Во-первых, исследователи ожидают получить, в определённой степени, детерминированное поведение трафика. Тем не менее, из-за некоторых компонентов, используемых в современных ЦП, таких как потоки, протоколы когерентности кэша и неизвестный заранее набор инструкций, имеет

место именно стохастическая структура. Наиболее простая стохастическая модель трафика – Пуассоновский процесс. Во-вторых, трафик на интерфейсе *L*2–*L*3 опровергает детерминированность.

На рис. 5 представлено распределение групп (серий) на примере алгоритма шифрования/дешифрования AES.

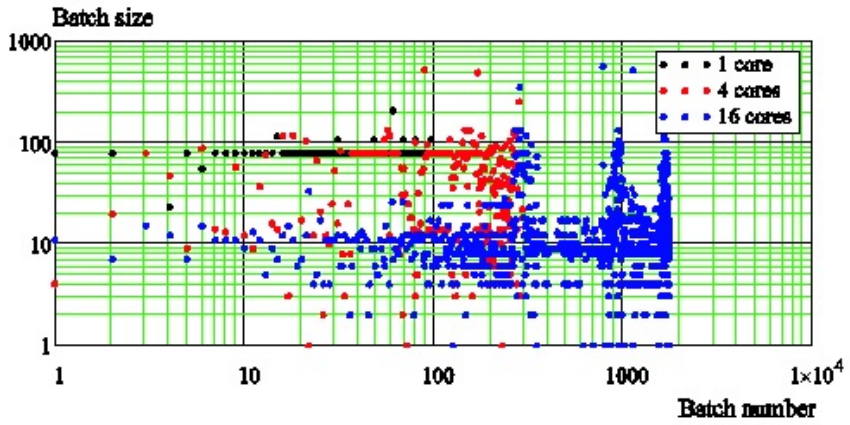


Рис. 5. Распределение групп трафика в зависимости от их размера

Нагрузка на интерфейс близка к детерминированной, но её «природа» по-прежнему является «групповой» или «пакетной» (*batch*-процесс) для одного активного ядра с размером группы в 80 временных интервалов. С увеличением числа активных ядер до 4 и далее до 16, наблюдается изменчивость в группах. Поэтому можно сделать вывод, что модель трафика является стохастической.

Для правильного выбора параметров общего интерфейса кэш-памяти необходимо использовать такую модель трафика, которая явно определяет «групповую» природу трафика. На рис. 6 представлены распределения групп трафика по частоте на примере алгоритма шифрования/дешифрования AES для 8 и 16 ядер.

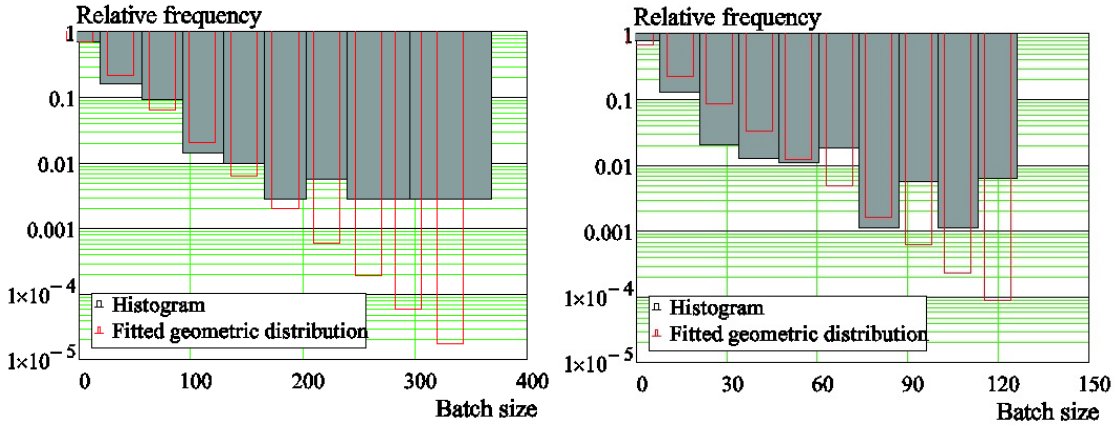


Рис. 6. Распределения групп трафика по частоте для 8 ядер (слева) и для 16 ядер (справа)

Гистограммы для системы с 8 и 16-ядрами имеют чётко наблюдаемое геометрически затухающее поведение.

## Выводы

Все приведённые методы оценки трафика в современных многоядерных процессорах общего назначения для определения модели трафика помогают подробно описать свойства трафика. Анализ микроархитектурного уровня при использовании документации, которая находится в открытом доступе, обеспечивает первый шаг к определению модели трафика. Определение набора алгоритмов и архитектурных решений поможет в решении применения необходимых инструментов.

Анализ функциональности ЦП и, в частности, протокола когерентности кэша позволит понять поведение различных подсистем и сделать вывод об уровне детализации для симулируемой модели. Микроархитектурный уровень симуляции при правильном взаимодействии всех компонентов, реализованных в современных ЦП, позволит понять «природу» трафика на различных интерфейсах ЦП и множество конкретных деталей. Абсолютные значения моделей трафика, полученные с использованием данного подхода, могут отклоняться от действительности из-за моделирования абстракций и неизвестных «ноу-хау» в реализованных алгоритмах.

Вычисления с использованием инструментов perf, Intel PCM позволяют понимать точное значение объёмов трафика на внутренних интерфейсах и настроить модель для дальнейшего использования методов симуляции. Измерения необходимы и для определения задержек между различными уровнями кэш-памяти.

Используя всю информацию, полученную тремя методами, можно вывести детальную модель трафика современных ЦП. Знание об архитектуре системы и о протоколе когерентности кэша позволяет определить модель трафика в случае увеличения количества ядер с помощью методов экстраполяции. В случае же добавления внутренних компонентов для коммуникации подсистем могут потребоваться значительные изменения в построении модели для симуляции, которые, в свою, очередь приведут к невозможности проведения экспериментов.

Как модель трафика, так и «стохастичность» изменяются с увеличение количества активных ядер. Поэтому были выбраны тесты с наименьшей и наибольшей сложностью вычислений, что только подчёркивает стохастическую модель трафика. Наконец, трафик имеет групповую структуру, т. е. разбит на серии, в которых периоды передачи данных чередуются с периодами пауз.

При разработке новых методов внутренней коммуникации ЦП должны быть приняты во внимание: поведение протоколов когерентности кэша и инструкций обработки процедур, а также структура трафика.

## Литература

1. Mujtaba H. Intel 2015–2016 Roadmap. URL: http://wccftech.com

2. Hammarlund P., Martinez A. J., Bajwa A. A., Hill D. L., Hallnor E., Jiang H., Dixon M. Derr M., Hunsaker M., Kumar R., Osborne R. B., Rajwar R., Singhal R., D'Sa R. Chappell R., Kaushik S., Chennupaty S., Jourdan S., Gunther S., Piazza T., Burton T. Haswell: The Fourth-Generation Intel Core Processor // IEEE Micro. 2014. Vol. 2. Iss. 2. pp. 6–20.

3. Binkert N. et al. The Gem5 Simulator // ACM SIGARCH Computer Architecture News. 2011. Vol. 39. Iss. 2. pp. 1–7.

4. Sanches D., Kozyrakis C. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems // 40th Annual International Symposium on Computer Architecture. 2013. pp. 475–486.

5. Molka D., Hackenberg D., Schone R., Mueller M. S. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System // 18th International Conference on Parallel Architectures and Compilation Techniques. 2009. pp. 78–86.

## References

1. Mujtaba, H. Intel 2015–2016 Roadmap. URL: http://wccftech.com

2. Hammarlund, P., Martinez, A. J., Bajwa A. A., Hill, D. L., Hallnor, E., Jiang, H., Dixon, M. Derr, M., Hunsaker, M., Kumar, R., Osborne, R. B., Rajwar, R., Singhal, R., D'Sa, R. Chappell, R., Kaushik, S., Chennupaty, S., Jourdan, S., Gunther, S., Piazza, T., Burton, T. Haswell: The Fourth-Generation Intel Core Processor // IEEE Micro. 2014. Vol. 2. Iss. 2. pp. 6–20.

3. Binkert, N. et al. The Gem5 Simulator // ACM SIGARCH Computer Architecture News. 2011. Vol. 39. Iss. 2. pp. 1–7.

4. Sanches, D., Kozyrakis, C. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems // 40th Annual International Symposium on Computer Architecture. 2013. pp. 475–486.

5. Molka, D., Hackenberg, D., Schone, R., Mueller, M. S. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System // 18th International Conference on Parallel Architectures and Compilation Techniques. 2009. pp. 78–86.

***Борунова Каролина Дмитриевна*** – аспирант, СПбГУТ, Санкт-Петербург, 193232, Российская Федерация, karolinadm@mail.ru

***Borunova Karolina*** – postgraduate, SPbSUT, St. Petersburg, 193232, Russian Federation, karolinadm@mail.ru