IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 18, 2016, accepted August 9, 2016, date of publication August 25, 2016, date of current version September 16, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2603169

Statistical Traffic Properties and Model Inference for Shared Cache Interface in Multi-Core CPUs

DMITRI MOLTCHANOV[1], ALEXANDER ANTONOV[2], ARKADY KLUCHEV[2], KAROLINA BORUNOVA[1], PAVEL KUSTAREV[2], VITALY PETROV[1], (Student Member, IEEE), YEVGENI KOUCHERYAVY[1], (Senior Member, IEEE), AND ALEXEY PLATUNOV[2]

[1]Nano Communications Center, Department of Electronics and Communications Engineering, Tampere University of Technology, FI-32720 Tampere, Finland
[2]ITMO University, Saint Petersburg 197101, Russia

Corresponding author: D. Moltchanov (dmitri.moltchanov@tut.fi)

This work was supported by the FiDiPro Program of Academy of Finland Nanocommunication Networks, 2012–2016, Government of Russian Federation, under Grant 074-U01.

ABSTRACT The general-purpose networks-on-chip (GP-NoC) has recently attracted the attention of the research and industry as a way to support the growing demands of computing systems. The design and the development of the communications and networking functions for such a large-scale versatile systems require knowledge of the traffic exchanged between the computing nodes. The object of the study in this paper is the last-level shared cache interface that is likely to be a traffic bottleneck in future GP-NoC architectures. First, using the direct measurements, we report on the stochastic traffic properties at large-scales, provide first two moments and distribution functions. Complementing measurements with fine-grained cycle-accurate CPU simulations, we then analyze the small-scale traffic behavior. We show that even for the simplest applications such as reading or writing of data, the nature of the traffic is stochastic, depends on the number of active cores, and irrespective of the application type, has an explicit batch structure. We further reveal that the batch sizes and inter-batch intervals can be well approximated by geometric distribution and the approximation becomes better when the number of active cores increases. These properties identify a simple arrival model that can be used in the analytical or simulation-based performance evaluation studies of the shared interface technologies in prospective NoCs.

INDEX TERMS Microprocessors, communication system traffic, cache memory, modeling.

I. INTRODUCTION

A NoC can be broadly defined as a unit featuring a number of information processing elements. Single-purpose NoCs (SP-NoCs) have been a topic of active research and development over the last decade. There are several examples of successful application of SP-NoC concept with graphical processing units being, possibly, the most widely-known to large audience. The reason for an extraordinary increase in performance of such systems that we have witnessed is mainly due to the nature of tasks allowing for their perfect parallelization, thus, leading to the simple structure of the processing elements and clear understanding of the requirements imposed on the communications subsystem and the cache coherence protocols.

The development of general-purpose central processing units (GP-CPUs) has also reached the level, where it became beneficial to scale the computational power horizontally by parallelizing the computations than to continue increasing the clock frequency. Addressing this issue, major manufactures, Intel and AMD, presented their dual-core GP-CPUs in 2005, spawning the era of multi-core CPUs. Starting from a simple integration of two computing nodes on a single chip and providing shared access to RAM, they have nowadays evolved to truly multi-core systems featuring 4, 6, 8 and beyond computing nodes on a single ship with deep integration between the components and dynamic threads redistribution between the cores [1], [2].

The application of the NoC concept to GP-CPUs is not straightforward as the computational tasks to be performed greatly vary in their specifics and the level of ''parallelization'' and, thus, may require intensive exchange of information between computational elements placing new requirements on the design of the CPU communication subsystem. Analyzing the recent literature on NoC design one could notice a significant gap between the approaches taken by computing and communications communities. The former

mainly targets design of efficient memory coherence protocols for GP-NoCs while the communications subsystem is assumed to be in place satisfying the intra-CPU traffic requirements. The communications community approaches the problem from the communications subsystem design perspective proposing new solutions targeting communications and networking mechanisms and paying less attention to the internal structure and the needs of a CPU including the question of memory synchronization. The latter is, however, of paramount importance as it often dictates the resulting performance of a chip, thus, providing extremely strict requirements on the loss and latency of the data delivery process.

The communications subsystem of modern GP-CPUs is based on simple topologies and communications mechanisms and is overprovisioned [1]–[3]. However, the increase in the number of cores on a single chip will inevitably require more efficient solutions. To understand the requirements imposed on the choice of the communications and networking technologies the knowledge of the traffic exchanged at the bottleneck interfaces of a GP-CPU is needed.

This work is a joint effort of computing and communications groups, attempting to bridge these communities together by identifying the crucial traffic properties at the shared cache interface of modern x86 GP-CPUs with hierarchal memory coherence subsystem for further development of efficient communications mechanisms. Carrying out both the measurement and simulation campaigns we analyze traffic properties of synthetic tests and real applications. Using direct measurements we demonstrate that the traffic is stable over long time periods possessing covariance stationarity and ergodicity. We obtain ''pure'' application traffic characteristics by getting rid of the background load and report on quantitative metrics of measured traces including moments and one-dimensional distributions. To make conclusions on stochastic traffic properties we further perform fine-grained cycle-accurate simulations using Gem5 CPU simulator. We show that even for the simplest applications such as sequential writing and reading, the traffic exchanged at the shared cache interface is stochastic, depends on the number of active cores and has clearly observable batch structure. We further study parameters of the traffic process showing that both batch sizes and inter-batch intervals can be well approximated by the memoryless geometric distribution. The latter is of special importance allowing to analyze the effect of prospective communications mechanisms using simple performance evaluation frameworks.

The paper is organized as follows. In Section II, we summarize the related work reminding the x86 GP-CPU architecture with hierarchical cache structure, describe traffic estimation methodologies and report on recent results in the area of interest. Further, in Section III, we introduce the system implemented for direct measurements of intra-CPU traffic and analyze basic statistical characteristics of traffic patterns including ergodicity and stationarity. Detailed investigation of the shared cache traffic is performed in Section IV, where we describe Gem5 experiments and report conclusions about the traffic properties. Conclusions are drawn in the last section.

II. BACKGROUND AND RELATED WORK

A. x86 GP-CPU ARCHITECTURE

Modern multi-core GP-CPUs (from now on, CPU) are an extremely complicated hardware consisting of hundreds of elements, such as computing cores, registers, cache memory for both instructions and data, cache controllers, power management subsystem, and buses connecting them together. The simplified vision of the modern CPU architecture is shown in Fig. 1, where only the data exchange network is sketched.

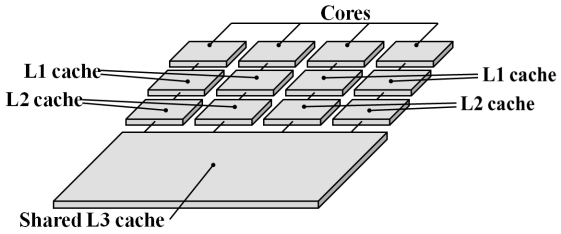

FIGURE 1. The conventional GP-CPU architecture with three cache levels.

To decrease the data and instructions access delay, the x86 architecture uses cache hierarchy. Three levels, L1/L2/L3, are commonly used [4], [15]. Each computing core has its own L1 and L2 caches, while the L3 cache is shared between all the cores. The cache latency of different levels depends on the cache size and grows from around $1-2$ns for 256KB L1 cache to $3-6$ns for 256-512KB L2 cache to $12-20$ns for 8-20MB L3 cache [3]. When cores are sending read/write requests to RAM the cache controllers use prediction algorithms to store the data that will likely be addressed in the future. The most recently accessed data are stored in L1/L2 caches. We distinguish between inclusive and exclusive hierarchical caches. In the former case the data contained in L1/L2 caches are always mirrored in L3 cache while in exclusive caches no data existing in L1/L2 caches are stored in L3 cache.

In addition to data buffering, L3 cache plays the role of the closest point along the way to the RAM, where the traffic from two different cores can meet. Thus, if core i is requesting the data core j is currently working with, the data transfer will be performed via L3 cache. To ensure the coherent view of data to all the cores cache coherence protocols are used. Even for such a simple hierarchal architecture providing memory coherence is a complex task with the number of states in the protocol state machine reaching few tens [13].

CPU vendors continue to evolve the current hierarchal cache architectures. There are various reasons behind this ranging from the miniaturization of the technological processes allowing to fit up to 8 full-scale x86 cores and 20MB of L3 cache on a chip [1] and promising extensions to 16 and 32 cores in the near future to development of new concepts extending the current architecture such as 3D stacked designs [14] allowing for efficient short-distance

TABLE 1. Comparison of cycle-accurate simulation environments.

Simulator	Libraries and instruments	Citation counts	Documentation	Downloads	Last update
SST	OpenMPI, Boost, DRAMSim2, Gem5	27	Wiki, doxygen	558	August 2015
Gem5	SWIG, protobuf	213	Wiki, slides, video guide, doxygen	5302	August 2015
ZSim	PIN	10	Installation guide	551	July 2015
MARSSx86	QEMU, PTLSim, SDL	41	FAQ, slides	473	April 2014

interconnects between cores, etc. Preserving the classic hierarchal cache design and possibly adding additional layers whenever needed keeps the latency at satisfactory level. However, increasing the number of cores the L3 cache might become a bottleneck for the whole system. There are a number of reasons for that ranging from the limited storage space in L3 cache (L3 is embedded on chip and takes around $30-40\%$ of chip space in x86 Intel processors [4], [15]) to the lack of space for multi-lane interfaces to the complexity of demultiplexing the access to L3 cache controller from a number of cores. The aim of this study is to understand the traffic dynamics at the shared cache interface providing the crucial building block for performance analysis of prospective efficient communications technologies.

B. INTRA-CPU TRAFFIC ANALYSIS APPROACHES

To study intra-CPU traffic one may choose three different approaches. According to the first, one could try to reveal internal traffic characteristics observing the processing logic of CPU subsystems. However, taking into account the complexity of modern CPU mechanisms including pipelining, replication and prediction, this approach does not appear feasible even for rather limited number of computational cores [15].

The second approach is based on measurements of traffic characteristics. In most modern CPUs there are no direct mechanisms to measure the traffic on an interface of interest. However, using Intel CPUs one can infer the amount of traffic indirectly, relying on the so-called ''performance counters'' provided starting from the Nehalem family of CPUs. Among others, these counters provide the information on the number of cache misses corresponding to a certain cache layer in an interval of a certain duration. There are several tools available for accessing these counters including perf [16] and Intel Performance Counter Monitor (PCM, [17]). An attempt to characterize the cache subsystem performance of Nehalem CPU family using these counters has been taken in [3]. However, as no timing information is provided by the tool, obtaining detailed traffic structure (e.g., time series of events) at the intra-CPU interfaces is impossible. Nevertheless, this approach allows to get distributions and moments of the traffic patterns at intra-CPU interfaces revealing their basic stochastic properties including stationarity and ergodicity as demonstrated below.

The approach that allows to characterize the traffic structure on internal interfaces in detail is cycle-accurate CPU simulation. There are a number of simulators supporting x86 architecture with MARSSx86 [18], Gem5 [19], zSim [20], and SST [21] being the most popular. The comparison of these modeling environments is shown in Table 1.

In our study we have implemented a typical x86 CPU architecture with appropriate cache subsystem in Gem5. Gem5 is a modular platform for computer-system architecture research supporting the Alpha, x86, ARM, MIPS, Power and SPARC ISAs. The simulator components are linked in Python environment. Gem5 supports full system cycle-accurate simulation, allows models of processor cores with different levels of detalization to be integrated, as well as coherence protocols and various DRAM models to be implemented. The project is actively evolving, has a rich ecosystem and is well documented. The shortcomings of Gem5 include low execution speed and limited scalability.

C. RELATED WORK

Over the last two decades, the question of analyzing intra-chip traffic characteristics in modern CPU and NoC systems has been addressed in several studies, within the computer architecture community, see, e.g., [5]–[10], among others. Since the focus of those investigations was mainly on refinement and development of cache coherence protocols they were performed using fine-grained cycle-accurate simulations. Furthermore, the authors mostly concentrated on short-scale behavior of traffic patterns. Nevertheless, a number of important fundings related to the intra-CPU traffic characteristics have been revealed so far. Particularly, most of the authors agree that intra-CPU traffic patterns are characterized by the batch structure, where long periods of interfaces' inactivity interchange with long durations of data transmissions. Synthetic traces demonstrating this behavior have been obtained and discussed in [8] and [9]. It has also been shown that the size of batches depends on the applications, number of cores and cache coherence protocols [10]. Opinions on the principal structure of the traffic vary with some studies advocating purely stochastic nature while others highlighting deterministic patterns.

In this paper we address the question of intra-CPU traffic characteristics for communications subsystem design. Thus, using the real measurement of operational CPU under different loads we first concentrate on principal properties of the traffic on the shared cache interface including its stochastic nature, ergodicity and stationarity that are critical for design of communications mechanisms. Further, using cycle-accurate simulations we will deepen the knowledge of descriptive statistics of intra-CPU traffic patterns confirming its batch structure even for simple write and read applications. In contrast to the referenced studies we further

proceed with traffic model inference for the shared cache inference.

III. MEASUREMENTS: BASIC TRAFFIC PROPERTIES

In this section, we describe our system for traffic measurements. We then illustrate basic properties of the traffic at the shared cache interface highlighting its stochastic nature, ergodicity and covariance stationarity. Finally, we report on statistical characteristics of obtained traces including moments and one-dimensional distributions.

A. DESCRIPTION OF EXPERIMENTS

In our experiments we have used the eight-cores Intel Core i7-5960X with Haswell architecture featuring 20Mb of L3 cache. We performed our experiments at the operating frequency 3.0GHz. The reference build of Linux Kubuntu 14.10 with kernel version 3.16 has been used. To access performance counters we have used PCM software [17].

Since measurements were performed on running operating system (OS) the measured data contain a certain amount of background load generated by OS services. On top of this, PCM itself impose additional load on CPU. Thus, there is a trade-off between the amount of reporting data per slot and the measurements accuracy. Using pilot experiments we have selected the reporting frequency of 5 measurements per second as the most suitable for our study.

TABLE 2. Selected applications and their specifics.

Applications	Details and specifics
Background	Default OS services are on
1B test	Sequential reading: byte-by-byte
64B test	Non-sequential reading: each 64th byte
Encryption	File encryption using AES-256 CBC
Compression	File compression using Tar+GZip
Kernel compilation	Compilation of Linux kernel
Video decoding	Decoding of HD video on CPU
Skype video call	Video and audio call via Skype

The applications we have considered and their details are summarized in Table 2. We specifically decided to rely real applications rather than on synthetic tests such as SPEC [11] or PARSEC [12], designed for CPU testing. We have also developed two artificial tests simulating the biggest and the smallest CPU loads, referred to as 1B and 64B. In both tests 1GB of memory was first allocated, and then reading from the array was initiated. In 1B test data are read sequentially, byte-by-byte, imitating the so-called ''good programming style''. As CPU reads data from the memory using cache lines of 64B in length, in 1B test we guarantee that the number of L2 misses is minimized imposing the lowest load on the shared cache interface. In 64B test we read each 64th byte ensuring that every time we address a new byte the shared cache interface is used. We call it ''bad programming style''. Note that both 1B and 64B test should generate deterministic load. However, the presence of background OS traffic and specifics of cache coherence protocol force us to handle their statistics using the general framework of stochastic processes. All tests lasted for 20 minutes resulting in 6000 samples. For all the tests we have performed two runs. All cores were kept operational simultaneously, running their instances of the test.

The load imposed at the shared cache interface was computed as follows. The recorded PCM data provides the values of counters, with ''L2MISS'' counter, describing the number of L2 cache misses. The shared cache interface is only used when there are no lines containing addressed byte in L1/L2 caches. In this case, the read request to L3 cache is sent. Processing this request, L3 cache controller finds the requested line in L3 or RAM and sends it back to L2 cache. The total amount of cache lines sent per time unit equals to the value of L2 misses happened during this period. In 64-bit CPU architecture, the cache line size is 64 bytes, while the read request length is 8 bytes. Thus, the total traffic on shared cache interface is

$$T = L2_m(8 + 64), \tag{1}$$

$L2_m$ is number of L2 misses per time unit.

We have performed statistical analysis for all considered applications. We report detailed results for four tests, 1B, 64B and Skype, as well as for OS background traffic. Selected 1B and 64B tests are very specific while the results of Skype tests are similar to those obtained for other applications.

B. TIME SERIES AND ANALYSIS METHODOLOGY

Fig. 2 illustrates the recorded time series of randomly chosen 50 successive samples for selected tests. We clearly notice the difference in the range of data: the load generated by 64B test is significantly higher than that of 1B and Skype tests. The load generated by Skype application is close to that of background traffic. Visual analysis of provided data allows to identify the general structure of traffic at shared cache cache interface. As we expected, the Skype trace demonstrates rather stochastic behavior. The background traffic is characterized by high peaks at non-random times reflecting the scheduled nature of OS processes. The 1B test trace is structurally similar to that of background traffic. Finally, the structure of 64B trace is different from that of 1B and background ones. This is expected as this test provides the highest load on the shared cache interface.

The visual analysis shows no indications of non-stationarity as the data look homogeneous with some deterministic behavior attributed to the background load. Thus, the data appear as a sum of two stochastic processes allowing to formulate the hypothesis consisting in that the background load is independent of the load of interest at the shared cache interface,

$$Z(t) = X(t) + Y(t) + W(t), \quad t = 0, 1, \ldots, \tag{2}$$

where $X(t)$, $Y(t)$, $W(t)$ are the load of interest, background load and load generated by the PCM tool, respectively. We can directly observe both $Z(t)$ and $Y(n) + W(t)$, although

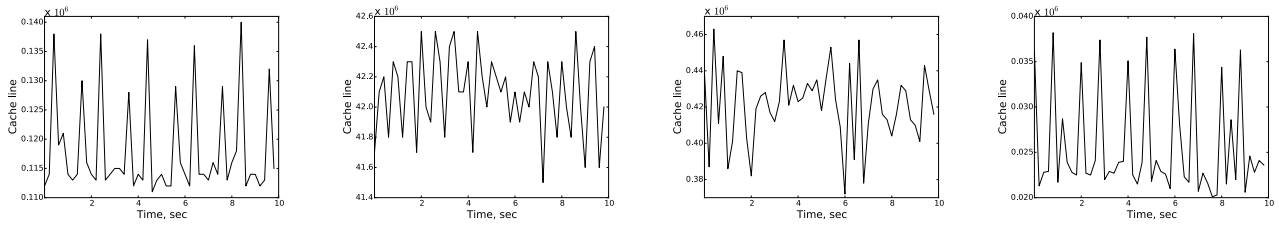

FIGURE 2. Times series of measurements. (a) 1B test. (b) 64B test. (c) Skype. (d) Background.

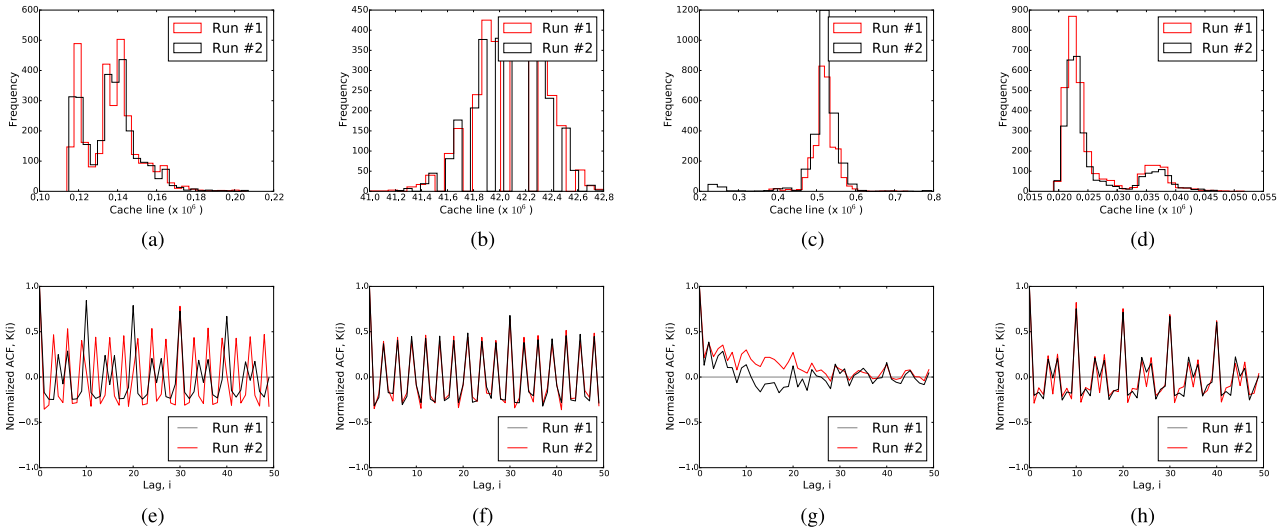

FIGURE 3. Ergodicity: comparison of histograms and NACFs of two traces. (a) 1B test. (b) 64B test. (c) Skype. (d) Background. (e) 1B test. (f) 64B test. (g) Skype. (h) Background.

not simultaneously. It is natural to expect that the contribution of $W(t)$ is low at the timescale of 0.2s.

Based on (2), we cannot construct the time-series of the process of interest, $X(t)$. However, if the involved processes are covariance stationary and ergodic we can directly measure $Z(t)$ and $Y(t) + W(t)$ and decide upon statistical characteristics of $X(t)$ indirectly.

C. ERGODICITY AND STATIONARITY

There are two major ways to study a stochastic process. One could infer statistical characteristics of a process using sections or realizations (sample paths). The latter is convenient but is only feasible when a process is ergodic. Ergodicity is an advantageous property of stochastic processes allowing to use only one sample path to decide upon all statistical characteristics of the process.

We have tested ergodicity of $Z(t)$ and $Y(t) + W(t)$ by comparing the histograms and normalized autocorrelation functions (NACF) of samples obtained in two different runs as shown in Fig. 3. As one may observe, the histograms match each other very well for all tests. The χ^2 test for homogeneity of samples conducted with the level of significance $\alpha = 0.05$ allows to accept the hypothesis that two samples were drawn from the same distribution. The closeness of NACFs for two runs indicates that the memory structure does not change from run to run. The special behavior of NACF functions is explained by the deterministic structure of synthetic tests and background load. For Skype test the structure inherent for a positively correlated stochastic process is observed.

Consider now stationarity of $Z(t)$ and $Y(t) + W(t)$. Recall that a process is called covariance stationary if its mean is constant over time and NACF depends on the time shift only. Fig. 4 shows statistical characteristics of considered processes including histograms and NACFs. For histograms we again applied the χ^2 homogeneity test with the level of significance $\alpha = 0.05$ and obtained the values less that the critical ones confirming that two samples are drawn from the same distribution. Thus, not only the mean but other moments are stable in time. The behavior of NACFs is more interesting. Although they depend on the time shift only implying that the considered processes are covariance stationary, the detailed behavior is different for different tests.

The NACFs of background traffic show very special behavior with peaks occurring at deterministic times separated by around 1s. They correspond to a certain process running in the background and requesting processing resources at these times. The same applies to two small peaks occurring in between two large ones. If an interval of 10 lags between

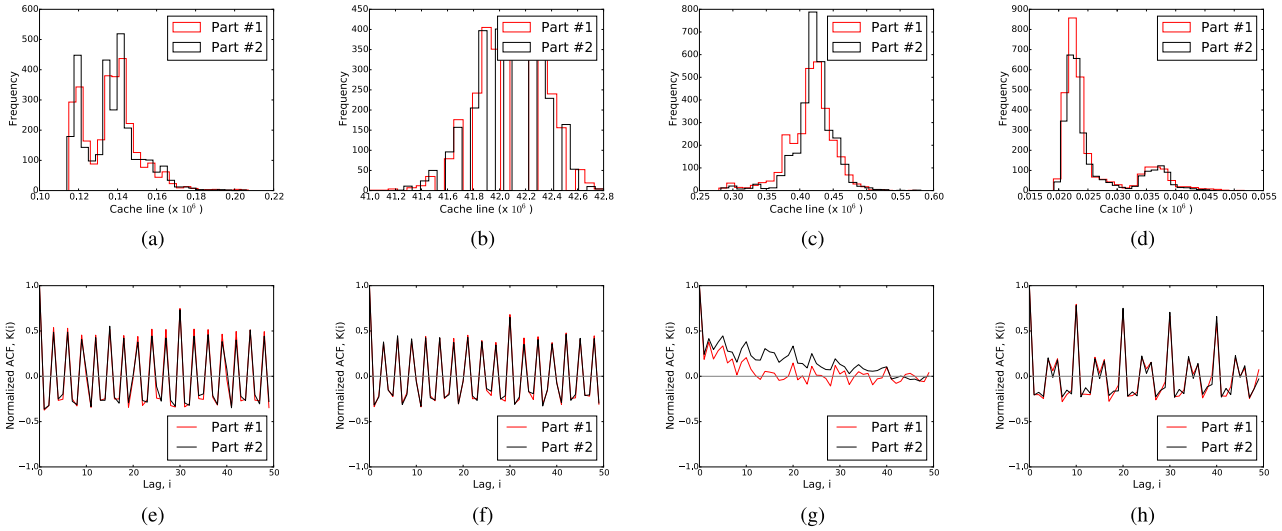

FIGURE 4. Stationarity: comparison of histograms and NACFs of two parts of the same trace. (a) 1B test. (b) 64B test. (c) Skype. (d) Background. (e) 1B test. (f) 64B test. (g) Skype. (h) Background.

TABLE 3. Statistical characteristics of "pure" traffic.

Test	$\hat{E}[X]$	$\hat{\sigma}^2[X]$	$\hat{\sigma}^2[X]/(\hat{E}[X])^2$
Background	0.129±0.006	0.0009	0.054
1B test	0.562±0.01	0.006	0.018
Skype video call	2.473±0.033	0.053	0.009
Encryption	5.413±0.25	3.034	0.104
Compression	9.091±0.324	5.108	0.062
Kernel compilation	9.564±0.411	8.243	0.090
Video decoding	10.989±0.114	0.635	0.005
64B test	176.552±10.96	5849.64	0.188

any two large peaks is observed one can identify 4 jumps upwards and 5 jumps downwards. These changes correspond to the execution of PCM tool itself as it logs data every 0.2s. In general, we see that the background traffic is a deterministic process that is observed together with the process of interest.

Observing NACFs of other traces we see the effect of deterministic background traffic translates to NACFs of compound processes. For 1B and 64B tests NACF functions repeat those of background traffic. This should not come as a surprise as 1B and 64B tests also generate almost deterministic load. For Skype we could also identify the same "peak" structure. However, in this case the nature of the application heavily contributes to the structure of NACF.

D. "PURE" TRAFFIC STATISTICS

The covariance stationarity and ergodicity of both $Y(t) + W(t)$ and $Z(t)$ implies that the process $X(t)$ is stationary and ergodic too. Furthermore, these properties indirectly confirm that the working hypothesis (2) is true as well. Thus, we can now determine statistical characteristics of $X(t)$ based on those of $Z(t)$ and $Y(t) + W(t)$. Table 3 shows interval estimates of the mean $\hat{E}[X]$, point estimates of standard deviation $\hat{\sigma}[X]$ and squared coefficient of variation, $\hat{c}^2 = \hat{\sigma}[X]/(\hat{E}[X])^2$, for all the types of applications considered.

IV. SIMULATIONS: DETAILED TRAFFIC STRUCTURE

The reported measurements provide information about the average traffic load at intra-CPU interfaces and general behavior of traffic patterns over long durations of time. However, the minimum time-scale of interest is too large to provide detailed understanding of traffic dynamics at the transactions level. In this section we complement our measurements campaign with cycle-accurate CPU simulations using Gem5, obtain traffic statistics at the transactions timescale and analyze them identifying critical traffic properties.

TABLE 4. Parameters of the simulated x86 system.

Parameter	Value
Gem5 basic model	DerivO3CPU
Clock frequency	1GHz
L1 data cache	64KB, 2ns latency
L1 instruction cache	32KB, 2ns latency
L2 data cache	2MB, 12ns latency
L3 data cache	16MB, 30ns latency
Number of cache levels	2, 3 with shared L3 cache
Number of cores	1, 2, 4, 8, 16

A. Gem5 MODEL AND MEASUREMENTS

The parameters of the simulated system are shown in Table 4. We considered a typical Intel x86 architecture including all the features and components implemented in Gem5. The chosen cache size and latencies are typical for modern CPUs. The cache subsystem was assumed to be inclusive. The model explicitly takes into account delays associated with information retrieval and emulates the pipelining capability. Systems with 1, 2, 4, 8 and 16 cores have been modeled. We emphasize

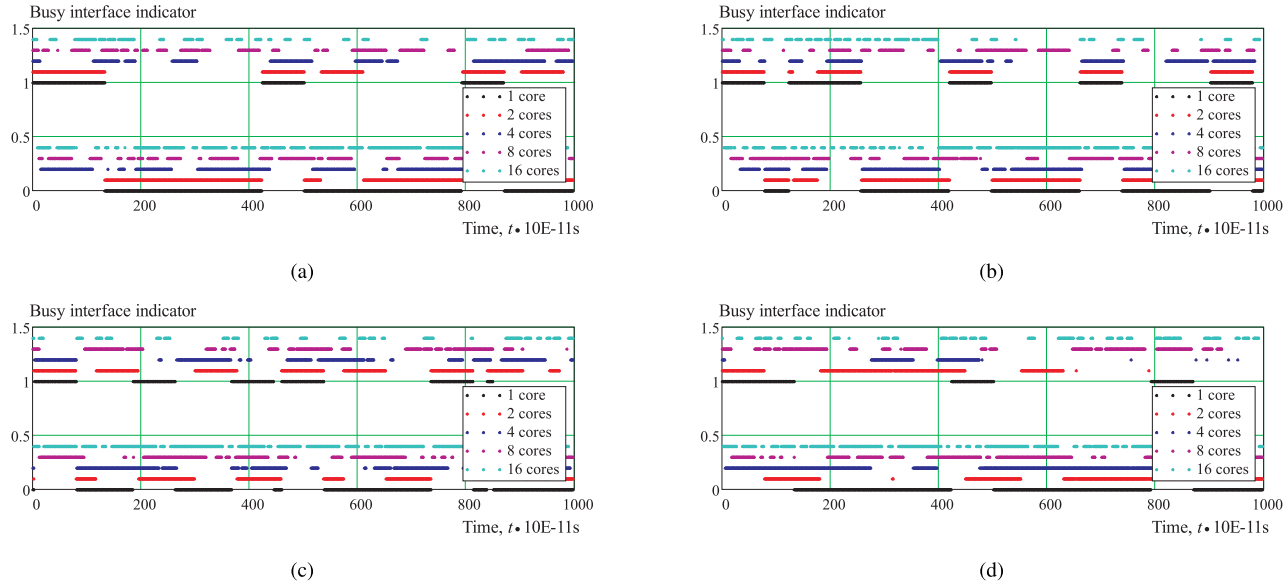

FIGURE 5. Traffic patterns at the shared cache interface. (a) Read test. (b) Write test. (c) Evklid test. (d) Factorial test.

that the clock frequency provides quantitative effect only and the obtained results can be scaled to any operational frequency.

TABLE 5. Brief description of testing programs.

Test	Brief description
read	Successive reading of 32 bits words out of 1000 bytes
write	Successive writing of 32 bits words up to 1000 bytes
bubble	Bubble sorting of 100 words array of 32 bits words
factorial	Computing $n!$ using recursive algorithm
euclid	Euclid algorithm for greatest common divisor
aes	Encryption/description of 16 bytes data using AES algorithm
zlib	Compressing 1000 bytes of data using zlib library

The selection of tests for simulation campaign is not a trivial task. First, full-cycle simulations are executed rather slowly preventing from emulating typical computer applications. Another reason restricting comprehensive tests is that the trace file generated by a simulator is extremely large even when a limited set of parameters is logged. Finally, a particular sequence of instructions executed by software is not known complicating interpretation of the results. Thus, to emulate a typical load at intra-CPU interfaces we have selected several representative tests covering various aspects of program code including simple reading, writing and sorting routines, more comprehensive recursive factorial estimation and Euclid's greatest common divisor algorithm involving divisions and multiplications, and most complex tests including AES encryption/decryption and compression using zlib. All the tests are listed in Table 5. Each test has been simulated for all considered number of cores. The number of simultaneously run tests were set to the number of operational cores. In overall, 70 tests have been performed.

For simulations we have used 64-bit Intel Core i7 equipped with 16GB of RAM running Linux Mint 17. The output of the simulation is stored in well-known ASCII value change dump (vcd) format. To visualize the data we have used GTKWave tool. To obtain time-series data the selected objects have been saved in "timing analyzer" (tim) format and then parsed.

B. STATISTICAL TRAFFIC ANALYSIS

The aggregated traffic pattern at the shared cache interface is demonstrated in Fig. 5. Visually observing the presented data we could state the following three hypotheses regarding the traffic nature. First, as one may observe, the traffic pattern changes when we increase the number of active cores. Secondly, the "stochasticity" of the traffic increases with the number of cores. We have specifically chosen tests having small and medium complexity to highlight that even those result in stochastic traffic pattern. Finally, the traffic has a clearly observable batch structure, where long transmission periods are interchanged with long pauses. We call them batch and inter-batch intervals, respectively.

To characterize the principal structure of the traffic pattern consider the test execution time illustrated in Fig. 6(a). As one may observe, enabling more cores results in more time required for execution of an individual test. This is explained by the effect of the cache coherence protocols and the common bus architecture of shared cache interface. Indeed, there should be additional time to arbitrate caching requests when the number of simultaneously operating cores increases. The increase is especially noticeable for low complex tests such as read and write as they require a plethora of shared cache transactions. The metric closely related to the execution time is the fraction of time the interface is busy, shown in Fig. 6(b). It can also be interpreted as the generated relative traffic volume and

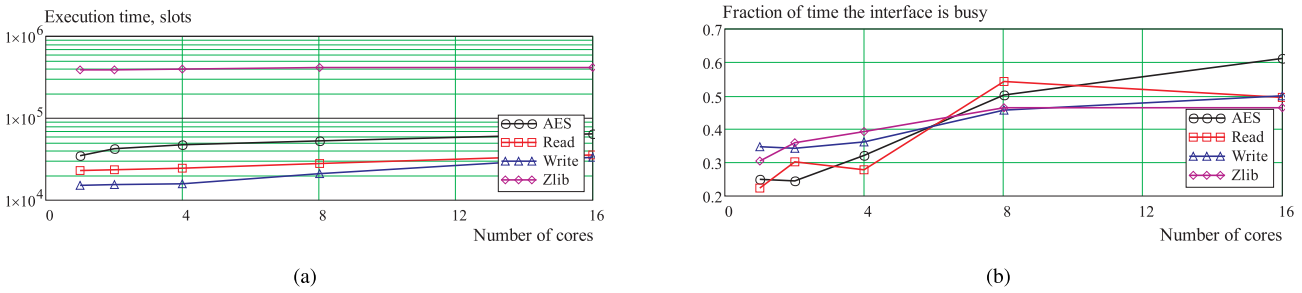

FIGURE 6. The effect of multiple cores: execution time and traffic volume. (a) Execution time. (b) Traffic volume.

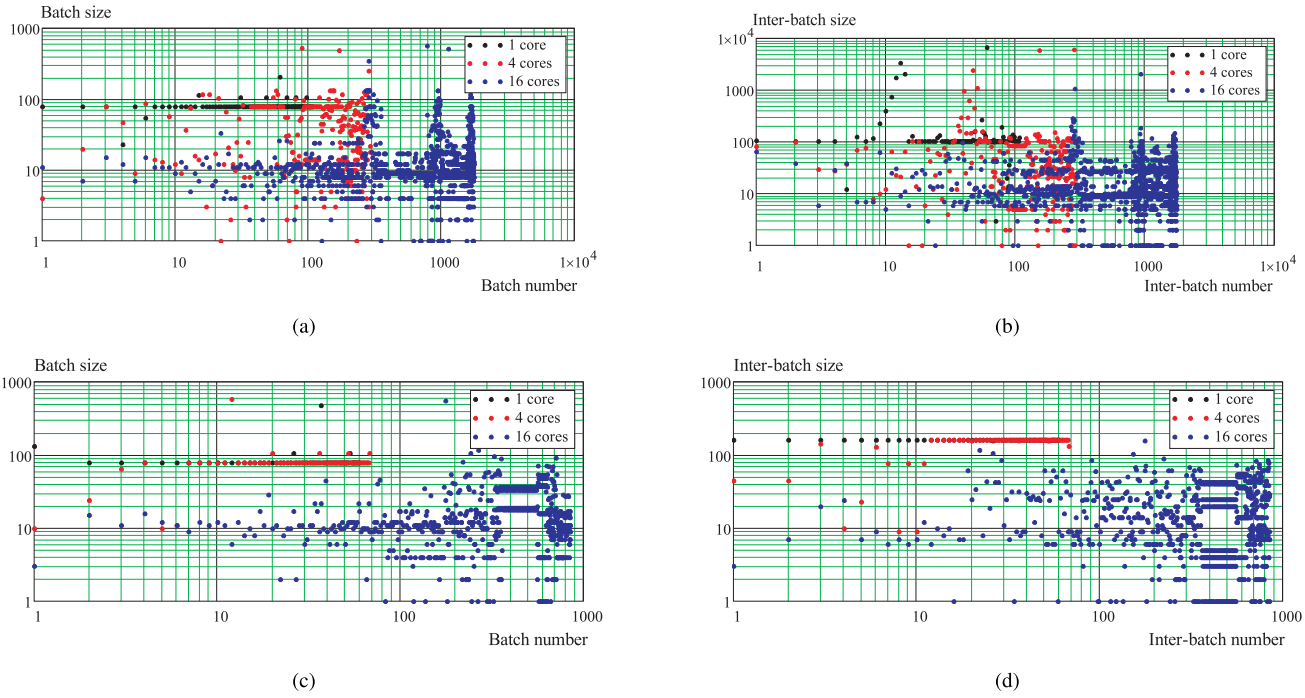

FIGURE 7. The patterns of batch and inter-batch intervals. (a) AES test, batches. (b) AES test, inter-batch intervals. (c) Write test, batches. (d) Write test, inter-batch intervals.

can be converted to absolute numbers using the rate of the interface. There is a trade-off behind this metric. Particularly, the higher number of active cores leads to longer execution time, thus, reducing the fraction of time the interface is busy. On the other hand, the amount of traffic increases as a result of more tests executed in parallel.

The abovementioned observations imply that in order to model the traffic at the shared cache interface we have to use batch traffic models. Let us now study the stochastic properties of batch sizes and inter-batch intervals. Fig. 7 shows time series of batch sizes and inter-batch intervals for simple write and complex AES tests. As one may observe, the load at the interface is close to deterministic but is still of batch nature for a single active core with most batch and inter-batch sizes having values of 80 and 108 time intervals, respectively. When the number of active cores increases to 4 and then to 6 we see much more variability in the batch and inter-batch observations. Also, notice that simple deterministic write test still produces mostly deterministic load even for 4 active cores while for the same number of cores the pattern appears stochastic for AES test.

Consider now the effect of the number of active cores on the point estimates of the moments of batch and inter-batch interval processes, shown in Fig. 8. As we expected, the mean duration of the batch and inter-batch intervals decreases as we enable more cores. The reason is that the load of the shared cache interface increases and transmission requests from/to L3 multiplexes on a common bus. Of special interest is the standard deviation characterizing variability of these intervals. As one may observe analyzing Fig. 8(c) and Fig. 8(d), the variability of the batch size first increases and then decreases for all considered tests. The increase is explained by the fact that tests are not well aligned in time while the decreases are due to multiplexing at the interface. One could also project this illustration to more cores expecting to have further reduction in variability as more cores are

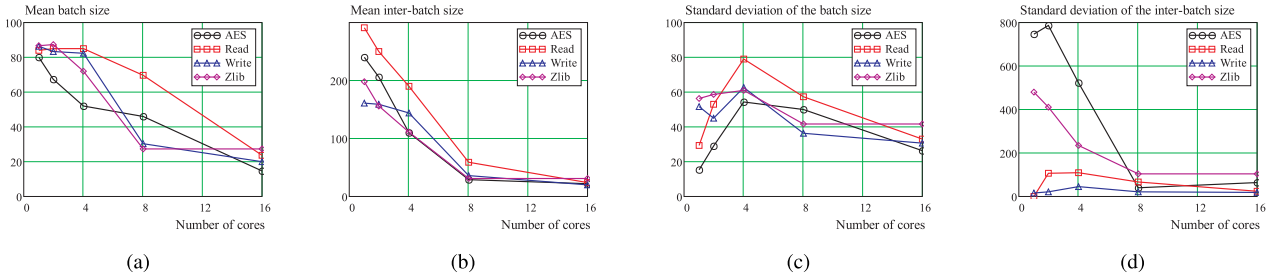

FIGURE 8. The moments of batch sizes and inter-batch intervals. (a) Mean batch size. (b) Mean inter-batch interval. (c) St. dev. of batch size. (d) St. dev. of inter-batch interval.

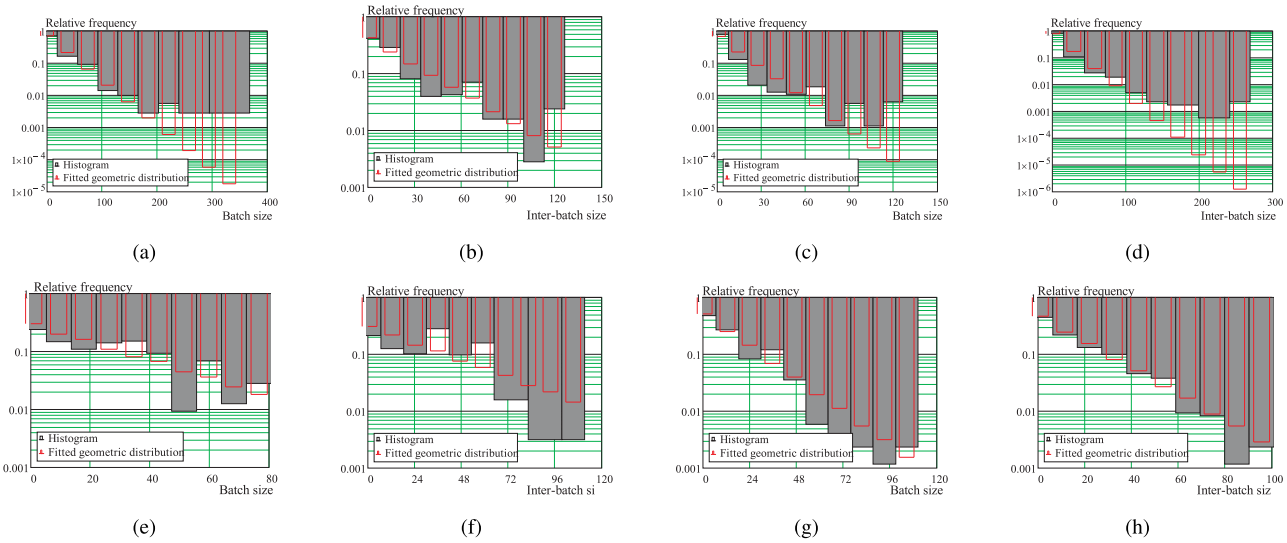

FIGURE 9. The histograms of batch sizes and inter-batch intervals and their approximations. (a) AES, 8 cores, batches. (b) AES, 8 cores, inter-batches. (c) AES, 16 cores, batches. (d) AES, 16 cores, inter-batches. (e) Write, 8 cores, batches. (f) Write, 8 cores, inter-batches. (g) Write, 16 cores, batches. (h) Write, 16 cores, inter-batches.

added to the system. The behavior of inter-batch intervals is even more interesting. For low complex applications such as write and read the variability remains at the same level except for the the small peak at 4 cores having the same nature as that of the batch size. However, for more complex applications requiring more processing resources, such as AES and Zlib the variability is extremely high for a single active core. The reason is that these tests are significantly more computationally intensive than simple read and write ones. Increasing the number of nodes to 8 and then to 16 greatly reduces variability and it becomes comparable to that of read and write tests.

To properly dimension the shared cache interface one needs to use a traffic model that explicitly captures the batch structure of the traffic. Let us now investigate the distributions of the batch and inter-batch intervals whose histograms of relative frequencies for AES and write tests are illustrated in Fig. 9. As one may observe, for 8 and 16 cores the histograms have clearly observable geometrically decaying behavior. The corresponding approximations highlight that geometric distribution may provide accurate first-order approximation for both batch sizes and inter-batch intervals. Note that the deviations are mostly attributed to low-weight probabilities. All these approximations passes χ^2 test with level of significance set of $\alpha = 0.1$, however, most of them do not pass with with $\alpha = 0.05$. As a result, one could study the performance of the shared cache interface in prospective GP-NoCs using the well-known polling queuing systems framework with deterministic service times and Geo^{Geo} arrival process (geometrically distributed inter-batch times batch sizes). The results for this system are immediately available, see, e.g., [22]. When the number of cores is small, e.g., 4 or less, the geometric distribution provides worse approximation.

V. CONCLUSIONS

Inspired by the recent interest in GP-NoC designs and the lack of traffic models for hierarchal cache coherence subsystems we analyzed the structure of the traffic at the shared cache interface. Using direct measurements we proved that the traffic generated by a wide range of applications is stochastic in nature, covariance stationary and ergodic. We also derived "pure" traffic statistics generated by individual applications. We also reported relevant

statistics including one-dimensional distributions and moments.

Using cycle-accurate CPU simulations performed in Gem5 we further showed that the shared interface traffic properties depend on the number of active cores. We revealed that the traffic structure has a clearly distinguishable batch structure. For systems featuring more than 4 cores the batch and inter-batch intervals can be accurately approximated by the geometric distribution. These conclusions hold even for the simplest tests such as successive reading or writing of data.

The detailed analysis of a traffic pattern at the shared cache interface allows to specify a representative yet simple traffic model as a function of the number of operational cores and the applications of interest. This enables the research community to start developing protocols for new efficient communications technologies and protocols in hierarchical cache based GP-NoC architectures. Particularly, the revealed traffic characteristics allow to represent the service process at the shared cache interface using a polling queuing system with Geo^{Geo} arrival process and deterministic service times. This model has been deeply studied in literature.

REFERENCES

[1] *Intel Core i7-5960x Processor Extreme Edition, Technical Specifications*, accessed on Jun. 7, 2015. [Online]. Available: http://ark.intel.com/products/82930/Intel-Core-i7-5960X-Processor-Extreme-Edition-20M-Cache-up-to-3_50-GHz

[2] *AMD FX Series Processors, Technical Specifications*, accessed on Jun. 7, 2015. [Online]. Available: http://www.amd.com/en-us/products/processors/desktop/fx

[3] D. Molka *et al.*, "Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system," in *Proc. 18th IEEE PACT*, Sep. 2009, pp. 261–270.

[4] H. Mujtaba. *Intel 2015-2016 Roadmap*, accessed on Mar. 1, 2015. [Online]. Available: http://wccftech.com

[5] L. Tedesco, A. Mello, L. Giacomet, N. Calazans, and F. Moraes, "Application driven traffic modeling for NoCs," in *Proc. 19th Annu. Symp. Integr. Circuits Syst. Design*, 2006, pp. 62–67.

[6] V. Soteriou, H. Wang, and L. Peh, "A statistical traffic model for on-chip interconnection networks," in *Proc. IEEE Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Sep. 2006, pp. 104–116.

[7] Y. Zhang, B. Ozisikyilmaz, G. Memik, J. Kim, and A. Choudhary, "Analyzing the impact of on-chip network traffic on program phases for CMPs," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2009, pp. 218–226.

[8] X. E. Chen and T. M. Aamodt, "Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs," *ACM Trans. Arch. Code Optim.*, vol. 8, no. 3, p. 10, 2011.

[9] G. Balakrishnan and Y. Solihin, "WEST: Cloning data cache behavior using stochastic traces," in *Proc. IEEE 18th Int. Symp. High Perform. Comput. Archit.*, Feb. 2012, pp. 1–12.

[10] M. Badr and N. E. Jerger, "SynFull: Synthetic traffic models capturing cache coherent behaviour," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, Jun. 2014, pp. 109–120.

[11] Standard Performance Evaluation Corporation. *The SPEC Benchmark, Software Tool*, accessed on Aug. 16, 2015. [Online]. Available: https://www.spec.org/benchmarks.html

[12] Princeton University. *The PARSEC Benchmark Suite, Software Tool*, accessed on: Aug. 11, 2015. [Online]. Available: http://parsec.cs.princeton.edu/

[13] J. Kadlec, "Simulation of cache hierarchy and the MESIF protocol," M.S. thesis, Faculty Inf. Technol., Czech Tech. Univ. Prague, Prague, Czech Republic, 2013.

[14] W. R. Davis *et al.*, "Demystifying 3D ICs: The pros and cons of going vertical," *IEEE Des. Test Comput.*, vol. 22, no. 6, pp. 498–510, Nov./Dec. 2014.

[15] P. Hammarlund, "Haswell: The fourth-generation Intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, Mar./Apr. 2014.

[16] *Perf, Software Tool*, accessed on Oct. 2, 2015. [Online]. Available: https://perf.wiki.kernel.org

[17] Intel Corp. *Intel Performance Counter Monitor, Software Tool*, accessed on Oct. 2, 2015. [Online]. Available: http://www.intel.com/software/pcm

[18] GNU Lic. *MARSSx86—Micro-Architectural and System Simulator for x86-Based Systems, Software Package*, accessed on Oct. 2, 2015. [Online]. Available: http://marss86.org/~marss86/

[19] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[20] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proc. ACM ISCA*, 2013, pp. 475–486.

[21] F. Rodrigues *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, 2011.

[22] H. Takagi and L. Kleinrock, "A tutorial on the analysis of polling systems," Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. 850005, 1985.

DMITRI MOLTCHANOV received the M.Sc. and Cand.Sc. degrees from the St. Petersburg State University of Telecommunications, Russia, in 2000 and 2002, respectively, and the Ph.D. degree from the Tampere University of Technology in 2006. He is a Senior Research Scientist with the Department of Electronics and Communications Engineering, Tampere University of Technology, Finland. He has authored over 100 publications. His research interests include performance evaluation and optimization issues of wired and wireless IP networks, Internet traffic dynamics, quality of user experience of real-time applications, and mmWave/terahertz communications systems. He serves as a TPC member in a number of international conferences.

ALEXANDER ANTONOV received the B.Sc. and M.Sc. degrees from ITMO University, Russia, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the Computer Science Department. He has authored several publications indexed in Scopus and Web of Science, and has taken part in a number of international conferences. His research interests include methods and technologies of embedded systems and systems-on-chip design, computer-aided design, and FPGA-based design.

ARKADY KLUCHEV received the Degree from ITMO University in 1993 as a Systems Engineer, and the Cand.Sc. degree in 1998. He is an Associate Professor with the Computer Science Department, ITMO University, Russia. He has authored over 40 publications. His research interests include embedded systems and real-time systems design.

KAROLINA BORUNOVA received the M.Sc. degree in 2015. She is currently pursuing the master's degree with the Faculty of Infocommunication Networks and Systems, St. Petersburg State Tampere University of Telecommunications, Russia. Her areas of interests are multicore processors, performance optimization, and Internet traffic.

PAVEL KUSTAREV received the M.Sc. and Cand.Sc. degrees from ITMO University, Russia, in 1995 and 1999, respectively. He is an Associate Professor with the Computer Science Department and the Head of Program Engineering and Computers Systems Faculty, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, ITMO University, Russia. His research interests include architectural design and analysis of systems-on-a-chip, networked embedded systems and cyber-physical systems-based on wired and wireless networks. He authored over 25 publications. He was the Head of the over ten R&D projects. He is a Supervisor for graduate and postgraduate students.

VITALY PETROV received the M.Sc. degree in information systems security from the Saint Petersburg State University of Aerospace Instrumentation, St Petersburg, Russia, in 2011, and the M.Sc. degree in communications engineering from the Tampere University of Technology, Tampere, Finland, in 2014, where he is currently pursuing the Ph.D. degree, focusing on enabling terahertz band communications for Beyond 5G wireless networks. He was a Visiting Scholar with the Georgia Institute of Technology, Atlanta, USA, in 2014 and a Strategic Intern with the Nokia Research Center, Helsinki, Finland, in 2012. His current research interests are in Internet-of-Things, terahertz band communications, nanonetworks, cryptology, and network security.

YEVGENI KOUCHERYAVY received the Ph.D. degree from the Tampere University of Technology, in 2004. He is a Professor and Laboratory Director with the Department of Electronics and Communications Engineering, Tampere University of Technology, Finland. He is the author of numerous publications in the field of advanced wired and wireless networking and communications. His current research interests include various aspects in heterogeneous wireless communication networks and systems, the Internet of Things and its standardization, and nanocommunications. He is an Associate Technical Editor of the *IEEE Communications Magazine* and Editor of the IEEE Communications Surveys and Tutorials.

ALEXEY PLATUNOV received the Degree from the Leningrad Institute of Fine Mechanics and Optics, Russia, and the Cand.Sc. and Dr.Sc. degrees in 1985 and 2010, respectively. Since 1987, he has been an Associate Professor, Professor, Chair of Computation Technologies, ITMO University, St-Petersburg. He has authored over 130 publications. His research interests include computer architectures, embedded systems and systems on chip, reconfigurable computing techniques, design methodologies and tools.