# A Load Factor and its Impact on the Performance of a Multicore System with Shared Memory

Dmytro Nedzelky[1], Maryna Derkach[1], Inna Skarga-Bandurova[2,1], Larisa Shumova[1], Svitlana Safonova[1], Volodymyr Kardashuk[1]

[1]Volodymyr Dahl East Ukrainian National University, 59-A Tsentralny Prospect, Severodonetsk, Ukraine, [nedzelsky, derkach, shumova, safonova, kardashuk]@snu.edu.ua, http://snu.edu.ua

[2]Oxford Brookes University, iskarga-bandurova@brookes.ac.uk, https://www.brookes.ac.uk

***Abstract*—The paper investigates the influence of the load factor of the shared memory on the efficiency of multicore systems. Typically, all cores serve threads of one program in parallel by the OpenMP programming technology or execute independent programs. There are no interactions between threads and independent programs, but conflicts can occur when accessing the shared memory. Models of program execution in one core and a multicore computer are developed, considering the probabilities of successful calls and service times of all levels of the shared memory subsystem. The load factor of the first level cache is determined through the ratio of the L1 cache load time to the total execution time of the program. The execution of various types of programs is simulated. A technique for the acceleration coefficient of a multicore computer based on the total load factor of the shared memory has been proposed. Based on this insight, we apply our model to determine the acceleration coefficient for 4-, 8-, 12- and 16-core systems for different combinations of system parameters.**

***Keywords— multi-core system; cache memory; shared memory; load factor; system performance***

## I. INTRODUCTION

Significant extension of data volumes and complexity of the programs run by computer systems raised the problem of increasing their performance, scalability and efficient utilization of the processor cores. Despite significant progress in hardware implementation, system performance analysis and load factor on the multicore system are still challenging tasks. Typically, the efficiency of a multicore system is defined using the acceleration coefficient of parallel program execution or through the set of programs when using $k$ cores. In reality, the acceleration achieved depends on many factors. In particular, not all of the program parts can be parallelized. Therefore, the program will remain consecutive fragments, the runtime of which will not depend on the number of cores used. Amdahl's law [1] considers the ideal situation when executing parallel programs in multicore systems without considering many other conditions.

The performance of multicore computers is investigated by testing real programs [2-5] or applying synthetic tests [6-8]. It was noted that until the specific threshold, the efficiency of multicore systems virtually does not change with increasing the core number, which contradicts with the single-core version, and after reaching the threshold, it becomes decreasing. The loss of performance can be significant. For example, according to both Intel and AMD [9], the utilization rate of one core can be decreased to several tenths; the more cores being involved, the lower the utilization rate of one core can be achieved. There are several reasons for this phenomenon, for example:

- decrease in the frequency of the cores due to thermal load;
- insufficiently efficient load balancing;
- idle kernels due to conflicts when accessing shared computer resources.

It can be argued that the conflicts for accessing the shared memory is one of the main factors for the decreasing utilization rate and overall performance of a multicore system.

Several recent studies have proposed different techniques to evaluate performance of multicore systems and parallel applications. In [9], the efficiency of multicore systems was investigated, taking into account the influence of the load on the total memory. To solve the problems in the shared memory and improve the multicore system performance, a memory scheduling strategy was proposed in [10]. Several papers were devoted to examinating different programming models in heterogenous multicore systems [11, 12]. A comprehensive survey on parallel performance problems on shared memory multicore systems is represented in [13]. However, a methodilogy for computing the intensity of the memory requests has not been detected. To the best of our knowledge, most of the previous studies on the efficiency of multicore systems lack a holistic methodology for determining the calls' generation intensity to the shared memory depending on the properties of the programs being executed with an acceptable error. Another aspect of this topic is the

efficiency of multicore computers through the acceleration factors and parameters of the computer cores. Therefore, the purpose of this paper is to study the intensity of calls generation to shared memory, the load factor of shared memory on the efficiency of a multicore computer, depending on the number of cores, properties executed by program kernels, parameters of kernels and shared memory.

The remainder of this paper is organized as follows. Section 2 presents a model of program execution in one-core system. Section 3 describes the proposed multicore system model and methodology. Section 4 presents the results achieved with proposed methodology. Finally, our conclusions and future work are described in Section 5.

## II. The Model of Program Execution in One Core

When a program is executed in one core, the following processes proceed in parallel: (1) all commands are read and prepared for the execution; (2) cache memories of three levels L1 (first-level cache), L2 (second-level cache), and L3 (third-level cache) are run and executed; (3) each request activates and runs functional units; and if necessary, requests are made to the computer shared memory. The entire execution time of some processes can be predicted with acceptable accuracy. For example, the busy times of the memory subsystem at all levels (cache memory and shared memory) can be achieved from the parameters of the core structure and levels of the memory subsystem, i.e. the number of memory access instructions in the program being executed, the cache structure memories of all levels, service times for requests at each level of the memory subsystem, the probability of successful hits in the cache memory of all levels.

However, the execution times of a number of processes cannot be predicted. In particular, without knowing the exact structure of the program being executed (sequential ordering of arbitrary commands, which commands depend on the results of previous commands, how many conditional transitions, how many commands of each kind, etc.), it is impossible to predict the degree of parallelism of the execution of commands in functional devices and the combination them with accesses to the memory subsystem [14]. The entire execution time of the program will be no less than the execution time of the longest stage of the parallel cycle.

When executing memory access instructions, the core accesses the L1 cache with the average time interval between adjacent requests. The request being successfully handled with probability $P_{L1D}$ is serviced in time $t_{L1D}$, and the command ends. In case of failure in the L1 cache with probability $(1 - P_{L1D})$, a request to the L2 cache is generated. If successfully handled with probability $P_{L2}$, the request is served by the L2 cache in time $t_{L2}$ and the command ends. In case of failure in the L2 cache with probability $(1 - P_{L1D}) \cdot (1 - P_{L2})$, a request is generated to the controller of the segment of the L3 cache. The request being successfully handled with probability $P_{L3}$, is served by the controller of the third-level cache segment in time $t_{L3}$ and the command ends. When the buffer of requests to the controller of the L3 cache segment is full, the core is blocked (stops generating requests). If there are no requests in the input buffers of the third-level cache segment controllers, they are idle. In case of failure in the L3 cache segment with probability $(1 - P_{L1D}) \cdot (1 - P_{L2}) \cdot (1 - P_{L3})$, a request is likely to be generated and placed in the input buffer of the memory controller. As long as the buffer of requests to the core memory controller is full, the controller of the L3 cache segment is also blocked (stops accepting new requests). Core memory executes requests in time $t_{OII}$. In the absence of the memory access request, it is idle.

The flow of requests from the core to the L1 cache is random. The randomness of the request flow depends largely from the type of program including the number of memory access instructions, the order in which they follow in the program, the execution time of the program, the parameters of the core and the entire computer, the properties of the locality of the program, etc. The actual form of the law of distribution in intervals between requests is unknown. From these perspectives, the following assumptions have been made.

The simplest kind of random process is the sequence of independent trials represented as a sum of n random processes without predominance of one of them. It can be described with the exponential distribution of time intervals between successive events from n = 4-5 [15]. Therefore, for our model, we also assumed that the process of accessing the first level cache is the simplest with the exponential distribution of time between requests.

It is also known that utilizing the exponential law of time distribution between successive requests is one of the "hard" modes of operation. Performance indicators, in this case, are minimal. Under other laws of distribution, they are seen to be higher and even better. This fact suggests using exponential law to achieve the lower boundaries of the system performance indicators.

Finally, we assumed that it is known: (1) $N_{LD}$ the number of memory access instructions in a specific program; (2) processing time for each cache level; (3) processing time in shared memory; (4) processing times by the communication subsystem; and (5) the likelihood of successful accesses to each cache level. Keeping this in mind, let us denote the full runtime of the program in the core by $T_{FULL}$. The busy times of the corresponding levels will be as follows

L1 cache busy time: $T_{L1D} = N_{LD} \cdot t_{L1D}$

L2 cache busy time: $T_{L2} = N_{LD} \cdot (1 - P_{L1D}) \cdot t_{L2}$

L3 cache busy time: $T_{L3} = N_{LD} \cdot (1 - P_{L1D}) \cdot (1 - P_{L2}) \cdot t_{L3}$

Core memory busy time:

$$T_{FULL} = N_{LD} \cdot (1 - P_{L1D}) \cdot (1 - P_{L2}) \cdot (1 - P_{L3}) \cdot t_{FULL}$$

Where $t_{LD1}$, $t_{L2}$, $t_{L3}$, $t_{FULL}$ denote service times of one request by the L1, L2, L3 cache-memories and core memory, respectively; $P_{LD1}$, $P_{L2}$, $P_{L3}$ are the probabilities of successful hits in the L1, L2, L3 cache memories, respectively.

We determine the utilization (load) of the first level cache by introducing $\alpha_{LD1}$ coefficient as ration of $T_{L1D}$ to $T_{FULL}$:

$$\alpha_{LD1} = T_{L1D} / T_{FULL}.$$

This coefficient enables the simulation of the execution of various types of programs. For example, $\alpha_{LD1} = 0.2$ means the L1 cache occupancy time is 0.2 of the total program execution time. At the same time, there is no need to make any assumptions about the composition of the commands in the program, the organization of the pipeline for processing commands in the core, the specific types of dependencies between the teams, and execution times of individual operations in functional devices.

The intensity of generating requests by one core to the first level cache

$$\lambda_{L1D} = \frac{N_{LD}}{T_{FULL}} = \frac{\alpha_{L1D} * N_{LD}}{t_{L1D} * N_{LD}} = \frac{\alpha_{L1D}}{t_{L1D}}$$

For the rest of the levels, requests generating intensity can be computed similarly.

The second level cache L2

$$\lambda_{L2} = \lambda_{L1D} * (1 - P_{L1D}) = \frac{\alpha_{L1D}}{t_{L1D}} * (1 - P_{L1D})$$

The third level cache

$$\lambda_{L3} = \lambda_{L2} * (1 - P_{L2}) = \frac{\alpha_{L1D}}{t_{L1D}} * (1 - P_{L1D}) * (1 - P_{L2})$$

The intencity for core memory

$$\lambda_{MEM} = \frac{\alpha_{L1D}}{t_{L1D}} * (1 - P_{L1D}) * (1 - P_{L2}) * (1 - P_{L3})$$

Then the load factor of memory by single core:

$$\rho^{i}_{MEM} = \alpha_{L1D} * (1 - P_{L1D}) * (1 - P_{L2}) * (1 - P_{l3}) * \frac{t_{MEM}}{t_{L1D}}$$

## III. THE MULTICORE SYSTEM MODEL

The further study of the effectiveness of a multicore system is based on the set of assumptions that (1) the computer has one processor chip with *k* cores; (2) buffers of sufficient size are implemented to achieve utilization coefficients close to the limit values; (3) all cores execute independent threads, which are independent and do not interact with each other, or they are threads of a well-parallelized program without interaction between them; (4) all processor cores use individual segments of the third-level cache memory, and there are no core accesses to "foreign" controllers of the L3 segments; (5) L3 cache is a combination device. This means that a new request can only be completed after completing the previous ones; (6) shared memory is also a combinational type device. A new request can be completed only after the completion of the previous request; (7) requests flow from the cores to the shared resource of a multicore computer (random access memory) is the simplest with the exponential distribution of intervals; (8) the requests flow served by shared memory is also the simplest with the exponential distribution law.

Considering the assumptions made, the functioning of a multicore computer can be represented as a two-phase queuing model. The first phase, consisting of k cores, generates memory requests. The total intensity of the request generation by k cores is

$$\lambda^{\Sigma}_{MEM} = \sum_{i=1}^{k} \lambda^{i}_{MEM} = \sum_{i=1}^{k} \frac{\alpha_{L1D}}{t_{L1D}} * (1 - P^{i}_{L1D}) * (1 - P^{i}_{L2}) * (1 - P^{i}_{L3})$$

In the case when all the cores perform identical (or close in composition) flows, we obtain

$$\lambda^{\Sigma}_{MEM} = k * \frac{\alpha_{L1D}}{t_{L1D}} * (1 - P_{L1D}) * (1 - P_{L2}) * (1 - P_{l3})$$

The second phase runs the shared memory with the intensity of the execution of the requests expressed as $\mu = 1 / t_{MEM}$. The core proceeds to generate a new request only after the generation of the previous request has been completed. When the buffer is full in the shared memory controller, the core is locked until free space appears in the buffer. The shared memory controller proceeds to execute a new request only after processing the previous request. If there are no commands ready for processing in the buffer and the core has not generated another request at this time interval, then the memory controller is idle.

The memory controller selects requests for execution from the buffer following the FIFO method. Fig. 1 shows a simplified block diagram of a multicore computer model.

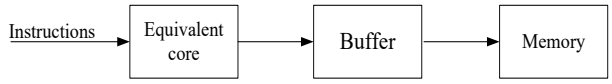


Figure 1. Simplified block diagram of a multicore computer model

We assume that at its most basic level, a multicore computer model with one equivalent core could be equal to a *k*-core model with buffer and memory since (1) the productivity of the phases of the requests generation are equal; (2) buffers have the same size; (3) memory performances are equal; (4) the method of choosing the requests for execution are the same; and (5) the conditions for blocking the phases of generating requests are the same.

Then the process of investigating the functioning of the model can be described by following five steps: (1) Determine the states of the model; (2) Compile the state graph of the model; (3) Compile the system of equations for the probabilities of the model states; (4) Solve the system of equations; (5) Investigate the indicators of the efficiency of the model functioning depending on the values of various parameters.

The coefficient for equivalent kernel utilization rate can be expressed as $H = \frac{1-\rho_{MEM}^{n+1}}{1-\rho_{MEM}^{n+2}}$ if $\rho_{MEM} \neq 1$ and $H = \frac{n+1}{n+2}$ if $\rho_{MEM} = 1$, where $\rho_{MEM}$ denotes the total load factor of the shared memory. Load factors of memory with one core and *k* cores

$$\rho_{MEM}^{i} = \alpha_{L1D} * (1 - P_{L1D}) * (1 - P_{L2}) * (1 - P_{l3}) * \frac{t_{MEM}}{t_{L1D}}$$

$$\rho_{MEM}^{\Sigma} = \sum_{i-1}^{k} \rho_{MEM}^{i} .$$

## IV. RESULTS

By using proposed coefficients, it is possible to simulate the execution of different types of programs. At the same time, this releases from making assumptions about the composition of the commands in the program, the organization of the command processing pipeline in the kernel, the specific types of dependencies between the commands, the times of execution of individual operations in functional devices that can be useful is some practical applications. The resulting values of the total load factors of memory for 4-, 8-, 12- and 16-core systems for different combinations of parameters are given in Table 1 and Table 2.

TABLE I. TOTAL LOAD FACTORS OF MEMORY FOR $t_{L1D}$ =4T; $t_{L2}$ =10T; $t_{L3}$ =30T; $t_{MEM}$ =150 T

| $\alpha_{LD1}$ | $P_{L1D}$ | $P_{L2}$ | $P_{L3}$ | $\rho_{MEM}^{\Sigma}$ | | | |
|---|---|---|---|---|---|---|---|
| | | | | k=4 | k=8 | k=12 | k=16 |
| 0.2 | 0.7 | 0.80 | 0.90 | 0.180 | 0.360 | 0.540 | 0.720 |
| | 0.8 | 0.90 | 0.97 | 0.018 | 0.036 | 0.054 | 0.072 |
| 0.3 | 0.7 | 0.80 | 0.90 | 0.270 | 0.540 | 0.810 | 1.080 |
| | 0.8 | 0.90 | 0.97 | 0.027 | 0.054 | 0.081 | 0.108 |
| 0.4 | 0.7 | 0.80 | 0.90 | 0.360 | 0.720 | 1.080 | 1.440 |
| | 0.8 | 0.90 | 0.97 | 0.036 | 0.072 | 0.108 | 0.144 |
| 0.5 | 0.7 | 0.80 | 0.90 | 0.450 | 0.900 | 1.350 | 1.800 |
| | 0.8 | 0.90 | 0.97 | 0.045 | 0.090 | 0.135 | 0.180 |
| 0.6 | 0.7 | 0.80 | 0.90 | 0.540 | 1.080 | 1.620 | 2.160 |
| | 0.8 | 0.9 | 0.97 | 0.054 | 0.108 | 0.162 | 0.216 |
| 0.7 | 0.7 | 0.80 | 0.90 | 0.630 | 1.260 | 1.890 | 2.520 |
| | 0.8 | 0.90 | 0.97 | 0.063 | 0.126 | 0.189 | 0.252 |
| 0.8 | 0.7 | 0.80 | 0.90 | 0.720 | 1.440 | 2.160 | 2.880 |
| | 0.8 | 0.90 | 0.97 | 0.072 | 0.144 | 0.216 | 0.288 |

TABLE II. TOTAL LOAD FACTORS OF MEMORY FOR $t_{L1D}$ =4T; $t_{L2}$ =12T; $t_{L3}$ =36T; $t_{MEM}$ =180T

| $\alpha_{LD1}$ | $P_{L1D}$ | $P_{L2}$ | $P_{L3}$ | $\rho_{MEM}^{\Sigma}$ | | | |
|---|---|---|---|---|---|---|---|
| | | | | k=4 | k=8 | k=12 | k=16 |
| 0.2 | 0.7 | 0.80 | 0.90 | 0.2160 | 0.4320 | 0.6480 | 0.8640 |
| | 0.8 | 0.90 | 0.97 | 0.0216 | 0.0432 | 0.0648 | 0.0864 |
| 0.3 | 0.7 | 0.80 | 0.90 | 0.3240 | 0.6480 | 0.9720 | 1.296 |
| | 0.8 | 0.90 | 0.97 | 0.0324 | 0.0648 | 0.0972 | 0.1296 |
| 0.4 | 0.7 | 0.80 | 0.9 | 0.4320 | 0.8640 | 1.2960 | 1.7280 |
| | 0.8 | 0.90 | 0.97 | 0.0432 | 0.0864 | 0.1296 | 0.1728 |
| 0.5 | 0.7 | 0.80 | 0.9 | 0.5400 | 1.0800 | 1.6200 | 2.1600 |
| | 0.8 | 0.90 | 0.97 | 0.0540 | 0.1080 | 0.1620 | 0.2160 |
| 0.6 | 0.7 | 0.80 | 0.90 | 0.6480 | 1.2960 | 1.9440 | 2.5930 |
| | 0.8 | 0.9 | 0.97 | 0.0648 | 0.1296 | 0.1944 | 0.2593 |
| 0.7 | 0.7 | 0.80 | 0.9 | 0.7560 | 1.5120 | 2.2680 | 3.0240 |
| | 0.8 | 0.90 | 0.97 | 0.0756 | 0.1512 | 0.2268 | 0.3024 |
| 0.8 | 0.7 | 0.80 | 0.9 | 0.8640 | 1.7280 | 2.5920 | 3.4560 |
| | 0.8 | 0.90 | 0.97 | 0.0864 | 0.1728 | 0.2592 | 0.3456 |

The results of the dependence of the acceleration coefficient S on the parameters of the program and the multicore system shown in tables 1 and 2 are shown in table 3.

TABLE III. SYSTEM ACCELERATION COEFFICIENT IF $P_{L1D}$=0.7; $P_{L2}$=0.8; $P_{L3}$=0.97.

| $\alpha_{LD1}$ | k | $\rho_{MEM}^{\Sigma}$ form table | S | $\rho_{MEM}^{\Sigma}$ form table | S |
|---|---|---|---|---|---|
| 0.2 | 2 | 0.09 | 2 | 0.108 | 2 |
| | 4 | 0.18 | 4 | 0.216 | 4 |
| | 8 | 0.36 | 8 | 0.432 | 8 |
| | 12 | 0.54 | 12 | 0.648 | 12 |
| | 16 | 0.72 | 16 | 0.864 | 16 |
| 0.3 | 2 | 0.135 | 2 | 0.162 | 2 |
| | 4 | 0.27 | 4 | 0.324 | 4 |
| | 8 | 0.54 | 8 | 0.648 | 8 |
| | 12 | 0.81 | 12 | 0.972 | 12 |
| | 16 | 1.08 | 14.8 | 1.296 | 12.3 |
| 0.4 | 2 | 0.18 | 2 | 0.216 | 2 |
| | 4 | 0.36 | 4 | 0.432 | 4 |
| | 8 | 0.72 | 8 | 0.864 | 8 |
| | 12 | 1.08 | 11.1 | 1.296 | 9.3 |
| | 16 | 1.44 | 11.1 | 1.728 | 9.3 |
| 0.5 | 2 | 0.225 | 2 | 0.270 | 2 |
| | 4 | 0.45 | 4 | 0.540 | 4 |
| | 8 | 0.90 | 8 | 1.080 | 7.4 |
| | 12 | 1.35 | 8.89 | 1.642 | 7.4 |
| | 16 | 1.80 | 8.89 | 2.160 | 7.4 |
| 0.6 | 2 | 0.27 | 2 | 0.324 | 2 |
| | 4 | 0.54 | 4 | 0.648 | 4 |
| | 8 | 1.08 | 7.41 | 1.296 | 6.2 |
| | 12 | 1.62 | 7.41 | 1.944 | 6.2 |
| | 16 | 2.16 | 7.41 | 2.592 | 6.2 |
| 0.7 | 2 | 0.315 | 2 | 0.378 | 2 |
| | 4 | 0.63 | 4 | 0.756 | 4 |
| | 8 | 1.26 | 6.35 | 1.512 | 5.3 |
| | 12 | 1.89 | 6.35 | 2.268 | 5.3 |
| | 16 | 2.52 | 6.35 | 3.024 | 5.3 |
| 0.8 | 2 | 0.36 | 2 | 0.432 | 2 |
| | 4 | 0.72 | 4 | 0.864 | 4 |
| | 8 | 1.44 | 5.56 | 1.728 | 4.6 |
| | 12 | 2.16 | 5.56 | 2.592 | 4.6 |
| | 16 | 2.88 | 5.56 | 3.456 | 4.6 |

From Table III we can see that acceleration coefficient S of a multicore computer can be determined as follows: $S = k$ if $\rho_{MEM}^{\Sigma} \leq 1$ and $S = k / \rho_{MEM}^{\Sigma}$ if $\rho_{MEM}^{\Sigma} \succ 1$

## V. CONCLUSIONS

The main challenge when using proposed technique is to determine the load factor of the 1L cache. The value of this coefficient is largely depends from the number of memory access instructions in the program and their relationship with the total number of instructions in the program, as well as the type of processing instructions, the presence of various information dependencies between the instructions and the parameters of the actuators.

For the specific heavy-duty programs, it is possible to identify the sections of the program that make the greatest contribution during execution, to write simplified code for implementing these sections in a pseudo-assembler. An analysis of these programs will make it possible to determine the structure of the program (the number of memory access instructions, the presence of information dependencies between the instructions, the type of data being processed). Knowing the parameters of the processor (latency of cache memories, their volumes, execution times of the main operations, frequency), memory of a specific multi-core computer, as well as the structure of programs, makes it possible to determine a specific range of values of the load coefficient of the cache of the first level and use the proposed method to determine the coefficient accelerating a multicore computing system when executing a specific program.

## REFERENCES


[1] Amdahl G.M. Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities, *Procceding AFIPS Conference*, Vol. 30, (Atlantic City, New Jersey, Apr. 18-20), AFIPS Press, Reston, Va., 1967, pp. 483-485.

[2] Pakhomov S. Applications for testing processors and PCs: Abbyy FineReader, WinRAR, Mathworks Matlab and Dassault SolidWorks with Flow Simulation. November 8, 2016 http://www.ixbt.com/cpu/cpu-testing-part7.shtml.

[3] Pakhomov S. Digital photo editing software as a test for processors and PC: Adobe Photoshop, Adobe Photoshop Lightroom and PhaseOne Capture One Pro. November 2, 2016. http://www.ixbt.com/cpu/cpu-testing-part6.shtml.

[4] Pakhomov S. Video editing software and video content creation as a test of processors and PC: Adobe Premiere Pro, Magix Vegas Pro, Magix Movie Edit Pro, Adobe After Effects and Photodex ProShow Producer. October 24, 2016. http://www.ixbt.com/cpu/cpu-testing-part5.shtml.

[5] Pakhomov S. Backup applications and FineReader for testing PC performance. March 11, 2018 http://www.ixbt.com/cpu/cpu-testing-part6.shtml.

[6] Khizhnyak N. Tests of Core i7-10700K, i5-10600K and i5-10600KF in Geekbench. 04/20/2020. Permanent URL: https://3dnews.ru/1008924

[7] Sozinov A. Tiger Lake smashed the Ryzen 7 4800U in the single-core test, but lost in the overall standings. 08/04/2020. Permanent URL: https://3dnews.ru/1017377

[8] Parovishnik V. AMD 3rd Gen Threadripper Zen 2 Sharkstooth 32-Core Beast "kills" all contenders in Benchmark Leak. 29-08-2019. Permanent URL: https://www.techpowerup.com

[9] IBM documentation: Factors that influence the performance of shared memory partitions https://www.ibm.com/docs/en/power8/8408-44E?topic=partitions-factors-that-influence-performance-shared-memory

[10] Fang, J., Wang, M. & Wei, Z. A memory scheduling strategy for eliminating memory access interference in heterogeneous system. *J Supercomput* 76, 3129–3154 (2020). https://doi.org/10.1007/s11227-019-03135-7

[11] Lai C., Shi X., Huang M. Efficient utilization of multi-core processors and many-core co-processors on supercomputer beacon for scalable geocomputation and geo-simulation over big earth data. *Big Earth Data* 2 (2018): 65 - 85.

[12] Lee J.H., Shi W., Gil J.M. (2018) Accelerated bulk memory operations on heterogeneous multi-core systems. *J Supercomput* 74(12):6898–6922.

[13] Atachiants, R., Doherty, G.J., & Gregg, D. (2016). Parallel Performance Problems on Shared-Memory Multicore Systems: Taxonomy and Observation. *IEEE Transactions on Software Engineering, 42*, 764-785.

[14] D. Nedzelskyi, M. Derkach, Y. Tatarchenko, S. Safonova, L. Shumova and V. Kardashuk, Research of Efficiency of Multi-core Computers with Shared Memory, *2019 7th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, 2019, pp. 111-114, doi: 10.1109/FiCloudW.2019.00032.

[15] Ross S. Introduction to Probability Models. *Acedemic Press*. 12 Ed., 2019, 842 p.