Indonesian Journal of Electrical Engineering and Computer Science
Vol. 34, No. 3, June 2024, pp. 1879~1892
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v34.i3.pp1879-1892

Random access memory page caching: a strategy for enhancing shared virtual memory multicomputer systems performance

Stepan Vyazigin[1], Madina Mansurova[1], Victor Malyshkin[2], Aygul Shaykhulova[1,3]
[1]Department of Artificial Intelligence and Big Data, Faculty of Information Technologies, Al-Farabi Kazakh National University, Almaty, Kazakhstan
[2]Department of Parallel Program Synthesis Laboratory, Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
[3]Department of Mechanical Engineering Technology, Ufa University of Science and Technology, Ufa, Russia

Article Info

Article history:

Received Nov 19, 2023
Revised Feb 13, 2024
Accepted Mar 4, 2024

Keywords:

Boolean
Combinatorial space
Data caching
Random access memory
Systems with shared virtual memory
Working set

ABSTRACT

This study examines a modified approach to optimizing the performance of support vector machine (SVM)-type multicomputer systems through a distinct type of caching method that allocates space in the random access memory (RAM) of a computing node for caching pages. The article extensively describes research on enhancing the performance of the SVM system through memory page caching in RAM at the hardware level by implementing the SVM system based on field-programmable gate arrays (FPGA). A systematic comparative evaluation highlights a discernible enhancement in system performance relative to systems not equipped with the revised caching algorithm. These findings could prove instrumental for subsequent studies focused on optimizing the performance of SVM systems, providing empirical data to inform future investigations and potential applications in multicomputer system performance enhancement.

Corresponding Author:

Madina Mansurova
Department of Artificial Intelligence and Big Data, Faculty of Information Technologies
Al-Farabi Kazakh National University
Almaty, 050000, Kazakhstan
Email: madina.mansurova@kaznu.edu.kz

1. INTRODUCTION

Recently, with an increase in the volume of data and the complexity of algorithms, computational performance has become one of the main criteria for the effectiveness of computing systems with virtual memory, for example the support vector machine (SVM) type [1], [2]. Virtual memory, a widely used memory management concept, extends beyond the conventional memory structure found in desktop PCs and multicomputer systems, including shared virtual memory (SVM) systems. Additionally, it acts as a bridge between the CPU and GPU, as illustrated in the compute unified device architecture (CUDA) [3].

Currently, one of the main areas of research in the field of improving the performance of SVM-type systems is restructuring [4]-[15] and refactoring [16]-[24] of programs. But there are other areas of research in this area, for example [25], [26]. Another approach using the cluster concept, based on Zhuravlev's approach to classification [27]-[30], was proposed in [31]-[34] and is associated with the construction of a special algebra over cluster algorithms. Then the main problem reduces to the solvability of the special operator equation. Our approach is based and developed from another idea [33] and also comes from the field of classical discrete extremal problems. Here, we can say, the approach of extremal problems is a very advantageous tool for optimizing or improving the behavior of systems [35]-[40] and is a natural step in the development of research.

It is also worth noting that one of the ways to increase performance is data caching. However, this method has both advantages and disadvantages. The advantages of this method include the speed of operation, and the disadvantages are the small cache size. Due to the development of technology in recent years, the speed of accessing RAM has become as close as possible to the speed of accessing the cache, and at the same time, the size of the memory in RAM is an order of magnitude larger. If earlier, due to the low speed of RAM, it was not effective to use it as a cache and RAM was used mainly for temporary data storage, then with an increase in the speed of processor access to RAM, it became possible to allocate additional space in RAM for caching and, as a result, the caching method has a new stage of development.

It is worth noting that in distributed virtual memory (SVM) systems, when data is accessed over the network, caching can be implemented at the node level, which can significantly speed up data access and improve overall system performance. However, to cache any data, storage space is required, and this article proposes using part of the RAM to cache virtual memory pages. Directly comparing the results of restructuring and refactoring with caching is quite difficult because these are completely different approaches, each of them has its own pros and cons, but caching is the main way to improve the performance of any system. Caching works no matter how well the program is made and can be used in conjunction with other approaches to improve performance. The advantage of caching in RAM on computing nodes is also influenced by the fact that RAM performance has been actively increasing in recent years, for example [41]-[44].

Currently, there is a wide range of caching algorithms, and this article presents research based on the working set strategy set), which consists of caching only those pages that are actively used by the application at a given time. This strategy can significantly reduce the amount of cached data and increase caching efficiency. Research results on the working set strategy for memory management were described in [45].

The main goal and distinctive feature of this work is to study the possibilities of using page caching in the RAM of computing nodes in multicomputer systems such as SVM using a working set strategy based on the mathematical model described in [46] and improved by us in [47], in order to increase the performance of this system as a whole, both in software and and at hardware levels. Further in this paper, briefly proposes a caching model, which will be discussed in section 2. Analysis and experimental results are depicted in section 3. Finally, section 4 draws the conclusions.

2. MATERIALS AND METHODS

Consider a program composed of n interacting blocks, denoted as $b_1, b_2, \dots, b_n$ (abbreviated 1,2,… ,n), distributed across p pages of virtual memory, labeled as $S_{g_1}, S_{g_2}, \dots, S_{g_p}$, which we'll simplify as $S_1, S_2, \dots, S_p$. The execution of this code can lead to issues stemming from a high occurrence of page faults, significantly impacting system performance. These issues arise from unclear code structure, resulting in diminished performance for both the code itself and the overall system. This undesirable behavior may also be attributed to code segments written by different authors belonging to different programming groups and created at various points in time. In one scenario, any reference made during code execution from the resident set must point to only one block among $\{b_1, b_2, \dots, b_n\}$, while in another scenario, this constraint may not be obligatory.

Let us denote $v_r$by the length of the rth page $r = 1,2, \dots, p$and $l_i$by the length of the block $i = 1,2, \dots, n$. This notation suggests that the system accommodates multi-dimensional page sizes. In this context, blocks denote various components of the code, such as subroutines, linear code segments, independent interacting programs, and application data blocks. Block distribution$b_1, b_2, \dots, b_n$ between pages $S_1, S_2, \dots, S_p$is predetermined and represented through a logical matrix $x = (x_{ri})_{p\times n}$, where the element is $x_{ri} = 1$if the block $i$belongs to the page $r$and $x_{ri} = 0$otherwise. Let us denote all such matrices by X. In Figure 1, blocks are displayed on the corresponding pages in the form of vertical columns with their numbers and lengths, for example, a page S_{g_1}contains blocks$b_{j_1 1}, b_{j_2 1}, \dots, b_{j_{\lambda_1} 1}$ with lengths$l_{j_1 1}, l_{j_2 1}, \dots, l_{j_{\lambda_1} 1}$ accordingly.

Recall that some variants of the problem of moving program blocks (code) across pages$S_1, S_2, \dots, S_p$ with the goal of minimizing the cost functional are known as NP -hard problems. The problem in our case has its own specifics and does not yet have a final solution, which is already quite enough for research. Therefore, the main impetus for research remains the fundamental aspect of the problem, which intensifies efforts and can become the main motivation for research.

Returning to our problem, it's important to remember that the random data, represented by D={θ}, as well as the permutation strategy, have an impact on the functional value. Suppose that after running the code with a specific value θ∈D, a reference line of blocks (pages) corresponding to it, denoted by θ, is available. In this paper, the permutation strategy selected is the working set (WS) strategy. The resident set of R pages in RAM at any given moment during program operation aligns with the working set of pages at that

moment, denoted by the commonly used notation W(k,t) [4]. We regard the block analogy of the working set as the control state (cs) of the code (program), utilizing q cs as the appropriate notation. In Figure 1 the control state q there isq=(i_1,i_2,…,i_(m(q))), wherei_j the number of the block (j=1,2,…,m(q)) that belongs to q, and any of them are marked as. In Figure 1 for some moment t, cs q=(i_1,i_5,i_6,i_7, 〖i_10,i_15,i〗 _(m(q))) marked with the symbol⊗ and means any block (or its number) that does not belong to, q, a belongs to the corresponding page of the resident setR and is present in RAM. In Figure 1 as a resident set R shows the working set R(q,x) that is generated by cs q and matrix. Of course, link to cs q to the block that is marked as ⊗ or ∘ in Figure 1, does not give a page fault, unlike references to blocks that are outside R(q,x). In any case, you need to correctly determine whether a page fault occurs or not.

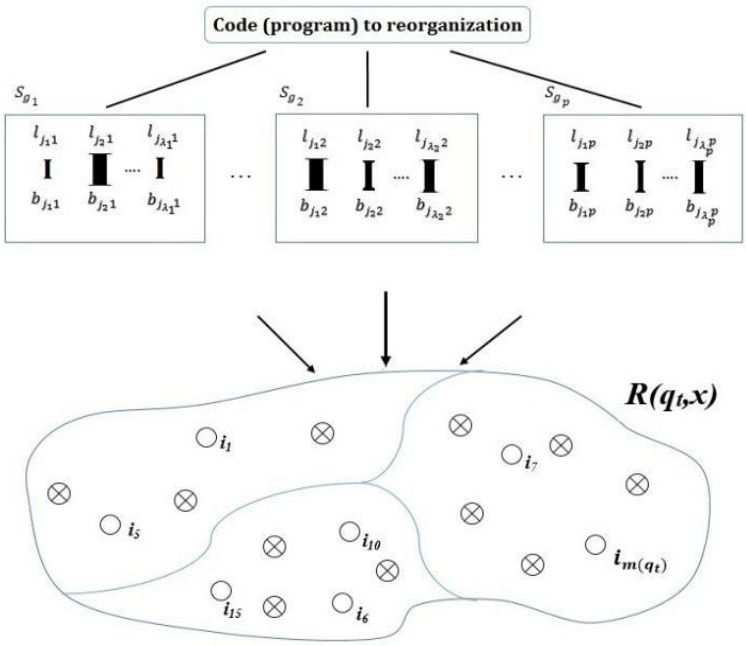


Figure 1. Code (program) reorganization scheme

It's crucial to emphasize that the matrix $x \in X$, is subject to inherent constraints (a)-(c), which we conceptually delineate in [47]. Functionals: For the primary optimization problem, the functional under consideration is the mathematical expectation of the page fault count per program (code) execution. As for an auxiliary objective, the functional represents the average page fault rate over h≥1 program (code) executions.

Note that this kind of functional is standard, for example, for stochastic programming and pattern recognition patterns and takes into account the random nature of the processes involved in the consideration, just like ours. In practical approaches, namely, in our understanding, the second functional is usually taken as a functional of the cost of code restructuring. However, an important question arises here: how$h \geq 1$ individual points in a data field span the entire infinite data field. In other words, the question is: how good is the solution to a problem with auxiliary functionality for the main task? Taking into account this kind of reasoning, we took as the functional of the main optimization problem the mathematical expectation of the number of page faults per one run of the program (code) instead of the average value functional, making an adjustment for the traditional interpretation of the formulation of the restructuring problem [16]-[24].

Constraint (a): this condition ensures that the combined length of blocks assigned to any page does not surpass the page's length.

Constraint (b): this restriction dictates that each program block (code) is assigned to only one page within the program (code).

Constraint (c): this requirement mandates that the cumulative length of any working set generated during program (code) execution does not surpass a predetermined system constant.

Keep in mind that these constraints (a)-(c) are established by the matrix $x = (x_{ri})_{p \times n}$ which dictates how blocks $b_1, b_2, \dots, b_n$ are distributed across pages $S_1, S_2, \dots, S_p$. Despite the wealth of literature dedicated to program restructuring, especially concerning our approach to the problem [4]-[15]. There is still no

definitive solution or appropriate reorganization model capable of achieving precise (optimal) solutions with the specified functionality.

2.1. Monitor the state of the program. Working set generated by control state q and matrix $x = (x_{ri})_{p\times n}$. Set of control states q

Let us recall some definitions. The working set W (k, t) with parameter k (window size) of program (code) pages in RAM at time t of program execution is the set of program pages that were accessed in the last k moments before t during program (code) execution. When discussing the concept of a working set, we typically refer to two related notations outlined by P. Denning: $W(t-\tau,t)$ with window size τ and $W(t-\tau,t)$ with parameter k. In the latter scenario, the integer k≥1 can also be seen as the window size. In the former case, the working set encompasses the program pages referenced within the interval [t- τ,t) of virtual time. In our context, we've adopted the W(k,t) option as the working set. Regarding the control state (cs) qt of the program at time t, we define it as a collection of program blocks referenced in the last k instances before time t. Consequently, the control state (cs) of a program (code) at time t essentially mirrors the working set of pages at that same time, but in terms of blocks.

It's crucial to highlight the significance of the Boolean matrix $x = (x_{ri})_{p\times n}$, which, as previously mentioned, plays a pivotal role in determining the program's structure, specifically how blocks $b_1, b_2, \dots, b_n$ are distributed across pages $S_1, S_2, \dots, S_p$. As established, this matrix x must adhere to constraints (a)-(c), and the collective set of all matrices of this type constitutes X. Ultimately, the objective is to identify an optimal matrix from the set X that delineates the most effective structure for the code (program) based on the aforementioned criteria.

2.1.1. Reference lines to pages and blocks. Control state q_t. Working set $R(q_t, x)$

While observing the reference line for a single program run and its associated frame, segmented into k cells below it, which progressively moves from left to right along the time axis t, it is essential to also consider Figure 2 simultaneously. In Figures 2 and 3, the instances t1, t2, t3 ,… , tγ represent time points sampled from t0 to tγ , which marks the conclusion of the run with random data θ ∈ D. In Figure 2, the labels St along the time axis t, where t=1, 2,3,… ,γ indicate the pages (or their identifiers) accessed during program execution for a specific θ ∈ D, with k=4 and fixed x ∈ X. For Figure 3 designations i(t_j), j=1,2,…,tγ denote block numbers corresponding to page numbers in Figure 2, accessed during program execution for the same θ ∈ D, with k = 4, and the same x ∈ X.

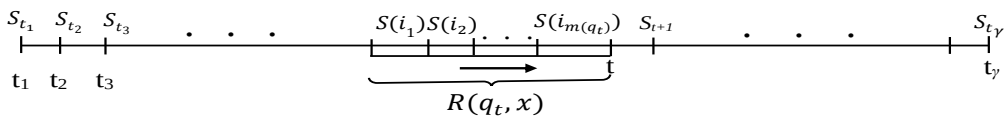


Figure 2. The line of reference to the program pages for one run of the program, where the working set R(q_t, x) is output, corresponds to W(4, t) with k=4 at t = t1, t2,…, tγ

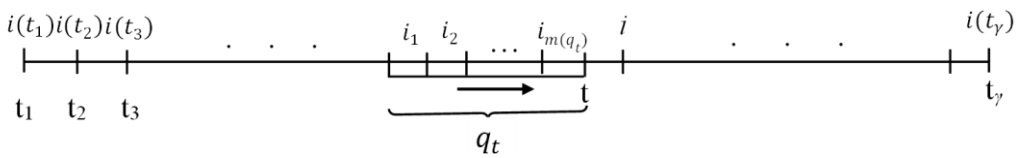


Figure 3. A string of program block references for one program run, where cs is printed for any t process q_t

The frame contents depicted in Figure 3 consist of blocks, which may include repetitions, constituting the control state (cs) $q_t$at time t. Meanwhile, the contents of the frame in Figure 2 comprise page numbers, also potentially with repetitions, which constitute the working set generated $R(q_t, x)$ by the coordinate system qt and the matrix $x = (x_{ri})_{p\times n}$ at time t. Each page $S(i_j)$in the frame contains a block i_j cs q_t, where $j = 1,2, \dots, m(q_t)$.

An important requirement for $R(q_t, x)$, is that each page within it must contain at least one block from the control state q_t. In our scenario, no other type of working set is considered. Hence, we have $\{S(i_1), S(i_2), \dots, S(i_{m(q_t)}\}$, as a multiset $R(q_t, x)$, but the actual working set $R(q_t, x)$ corresponding to it

should not include duplicate pages. Let's denote $R(q,x)$ as the working set without page repetitions corresponding to $R(q_t,x)$. Note that this conversion is easy to do by eliminating the repetition in qt to go from $R(q_t,x)$to $R(q,x)$. Based on Figure 3 it is easy to see that the next cs . q_{t+1} forms like $q_t \cup \{i\}$,i.e. $q_{t+1} = \{i_2, i_3, \dots, i_{m(q_t)}, i\}$. T such an event, i.e. link from cs q_t per block i which we can denote as $q_t \to i$ and number$i$ becomes available towards the end of cs processing q_t.

As shown in Figure 4, which contains a string of references to program at certain times, at the initial moment t_0 we do not have a line of links to pages and blocks, and as a result we have an empty frame. Despite this, for our convenience we have written down the block numbers in advance $i(t_1)$, $i(t_2)$, $i(t_3)$ above the t axis as shown in Figure 4(a), which will appear at the appropriate moments. Note that the block $i(t_1)$ is accessed just before the frame is reached $i(t_1)$ and instantly the number $i(t_1)$ falls into the rightmost cell of the frame as shown in Figure 4(b) and the same thing happens at time t_2 as shown in Figure 4(c).

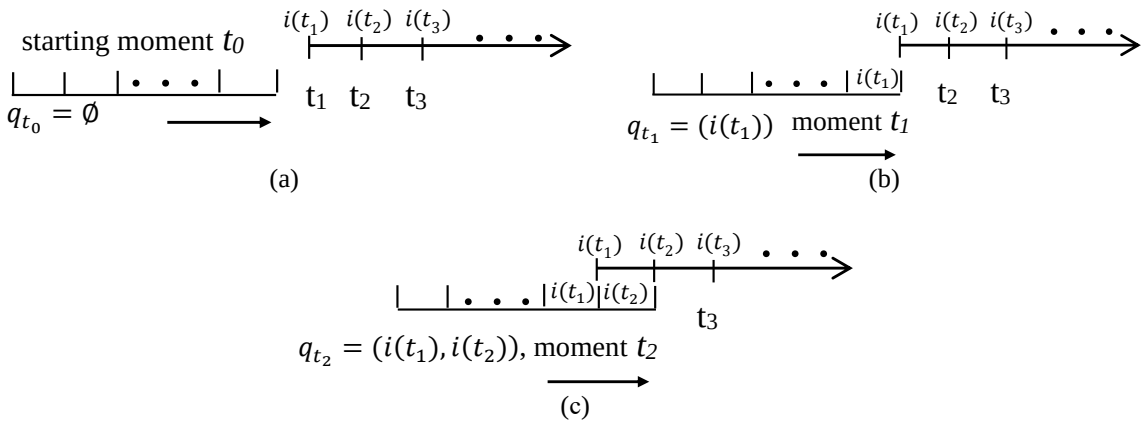


Figure 4. String of references to program at time: (a) t =0, (b) t = 1, and (c) t=2

Thus, as we see, the content of the frame at any moment t with repetitions of block numbers coincides with the control state (cs) of the program at moment t, the designation for it $q_t = \{i_1, i_2, \dots, i_{m(q_t)}\}$ as shown in Figure 3, in contrast to the corresponding designation for cs without repetitions, $q = (i_1, i_2, \dots, i_{m(q)})$ i.e. In addition, we can omit the index t q_t because for us there is no difference between $\{i_1, i_2, i_3\}$ and $\{i_2, i_1, i_3\}$ or $\{i_3, i_1, i_2\}$. And instead of all of them we will write an ordered entry (i_1, i_2, i_3), where $i_1 < i_2 < i_3$. And also for cs $q = (i_1, i_2, \dots, i_{m(q)})$ there is a relation: $i_1 < i_2 < \dots < i_{m(q)}$, where in there $q$are no longer blocked repetitions. By moving the frame along the chain of links to the blocks from left to right, we get the cs part. But it is possible that most of them are multisets, and we consider both of them, i.e. a multiset and its corresponding set without repetitions, as one and the same set. Despite this, for q_t, in contrast to q, a separate clarification of the notation has been introduced, namely, $q_t = \{i_1, i_2, \dots, i_{m(q_t)}\}$ it is treated q_t as a multiset. Having excluded from the found q_t repetitions of block numbers and then, if q_t new, then this one checks the comparison path q_t with q from the set $Q = \{q\}$ formed at the current moment t, q_t must include in it Q as new q. Thus, in the order of a new run of the program (code), little by little we enrich a situation where the set Q does not change. And finally, the set $Q = \{q\}$ consists of different blocks q,, where none of them q have block repetitions and Q does not change even after $\lambda \geq 1$ additional runs.

Here in Figure 5 set $\left\{q_\theta^1, q_\theta^2, \dots, q_\theta^{\mu(\theta)}\right\}$ represents a multiset of control states obtained during a single execution of the code, each corresponding to a distinct $\theta \in D$, and any new state, without repetition, is added to the set Q during subsequent runs, gradually approaching the stable condition depicted as "Is Q stable?" in Figure 5. When answered affirmatively for the first time, it signifies that Q was unstable in previous iterations but has now stabilized in the current iteration, exhibiting no deviation from the previous iteration's Q. If υ is less than or equal to λ, the process follows the top line, updating θ and continuing with the established pattern. The set J consists of blocks labeled with numbers i that emerge during the transition q_t→i as the frame progresses along the time axis in Figure 3. Following at least λ additional runs and reassigning new numbers from 1 to n to elements from J, we form a set of blocks requiring restructuring, which is suitable for further investigation.

However, a question emerges: what if, even after conducting λ or more additional runs, a new control state emerges? This indicates that the initial choice of λ was overly optimistic and necessitates

re-evaluating λ, leading to a requirement to restart the process. The scenario outlined above pertains to both the primary and auxiliary optimization problems. However, in the auxiliary problem, the set J is disregarded. It's important to highlight that in the auxiliary problem, the set J is predetermined.

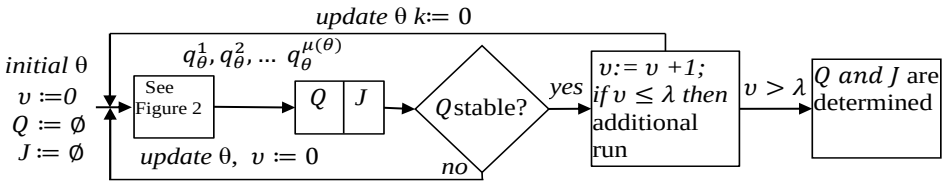


Figure 5. Q and J generation kits

2.1.2. Correlations between cs q_t and cs q_{t+1}

Here cs q_{t+1} as shown in Figure 3 is formed from a subset q_t, including the case itself q_t, which must be connected with $\{i\}$ and then the following relations hold: $q_{t+1} \subset q_t$ or $q_t \subset q_{t+1}$, or $q_{t+1} = q_t$. Indeed, cs $q_t = \{i_1, i_2, \dots, i_{m(q_t)}\}$ and $q_{t+1} = \{i_2, i_3, \dots, i_{m(q_t)}, i\}$ (see middle fragment of Figure 3), then we have several steps to continue:

Step I. If $i_1 \in q_{t+1} = \{i_2, i_3, \dots, i_{m(q_t)}, i\}$ and $i \notin \{i_2, i_3, \dots, i_{m(q_t)}\}$, then $q_t \subset q_{t+1}$, as shown in Figure 6(a).

Next situation

Step II. if $i_1 \in q_{t+1}$ and $i \in \{i_2, i_3, \dots, i_{m(q_t)}\}$ then $q_t = q_{t+1}$ as shown in Figure 6(b).

And further,

Step III. if $i_1 \notin \{i_2, i_3, \dots, i_{m(q_t)}\}$ and $i \notin \{i_1, i_2, \dots, i_{m(q_t)}\}$ then $q_t \nsubseteq q_{t+1}$ as shown in Figure 6(c). Here $q_t = \{i_1, i_2, \dots, i_{m(q_t)}\}$ and $q_{t+1} = \{i_2, i_3, \dots, i_{m(q_t)}, i\}$

Step IV. If $i_1 \notin \{i_2, i_3, \dots, i_{m(q_t)}\}$ and $i \in \{i_2, i_3, \dots, i_{m(q_t)}\}$ then $q_{t+1} \subset q_t$ as shown in Figure 6(d), where $q_t = \{i_1, i_2, \dots, i_{m(q_t)}\}, q_{t+1} = \{i_2, i_3, \dots, i_{m(q_t)}\}$

Steps I, II, IV are standard, but for step III we can assume that while the intermediate node q is going through the process, the time instant is t+1/2 as shown in Figure 6(c). In fragments of Figure 6 one can notice an arc (q_t, q_{t+1}), and an arc $(q_t, q_{t+1/2})$ and an arc $(q_{t+1/2}, q_t)$, each of which has a weight $(\pm i, \beta)$ where $\beta \in \{0,1\}$. Here $\pm i$ this refers to the movement up or down, which either enriches the cs q_t as shown in Figure 6(a), with i moving up or depletes cs q_t as shown in Figure 6(b), by i_1 (downward movement) or sequentially both of them as shown in Figure 6(c). Number i without a sign (Figure 6(d)) means that $q_t = q_{t+1}$ the situation β parameter equal to 1 or 0 and refers to the event when a page fault occurs or not on the corresponding transition from cs q_t to cs q_{t+1}.

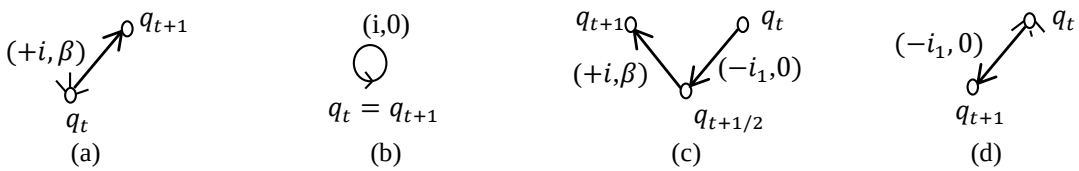


Figure 6. Options for movement in bipolar combinatorial space; (a) step I, (b) step II, (c) step III, and (d) step IV

2.1.3. Final notices for defining a set of control states Q

After conducting multiple executions of the program with various values of θ from the set D and repeated determination of cs q_t, once we encounter a situation where the set Q remains unchanged, and after at least λ≥1 additional executions where the set Q stays the same, we will consider the set Q to be defined. Initially, let's designate a special cs. $q_0 = \emptyset$ for the set Q, representing the starting point of the process. This initial state q_0, can also occur later if the program is unexpectedly evicted from main memory and later reactivated, essentially starting from scratch (cold start). Another scenario is a warm start, where the system ensures that the computing process, including control state, is restored to the state just before the program was evicted, allowing it to resume from where it left off without being affected by timeouts in secondary memory.

3. RESULTS AND DISCUSSION

SVM system built on 10 field-programmable gate arrays (FPGAs) forming a multi-computer system of the SVM type based on [48]. To study the effectiveness of caching in this system, the same memory access algorithms were run, which are the worst possible options for any type of caching. Namely, access to a random memory address. It is worth noting that the FPGA implements a high-frequency timer counter that measures the time it takes to receive the desired page. This approach allows you to obtain the time of not only medium and long-time page faults, but also short ones. Also, to minimize the error in measurements, a logging module similar to the methods presented in [49] was implemented on each of the FPGAs, the task of which was only to output data via USB-TTL to a separate computer. This data exchange method is widely used in many works, for example in [50].

The hardware part of this system is the DE2-115 development boards from Terasic, which include 2 network cards and all the necessary components, which made it possible to connect the devices into a local network using a ring topology. Conventionally, research on an FPGA system can be divided into 2 types as shown in Figure 7. Without RAM overflow and with RAM overflow of the computing node. In the first case, it is assumed that we have a reserve of memory in RAM on the computing nodes, which is not used by the system and is allocated for caching. And in the second case, memory is allocated by reducing the data currently located on the computing node.

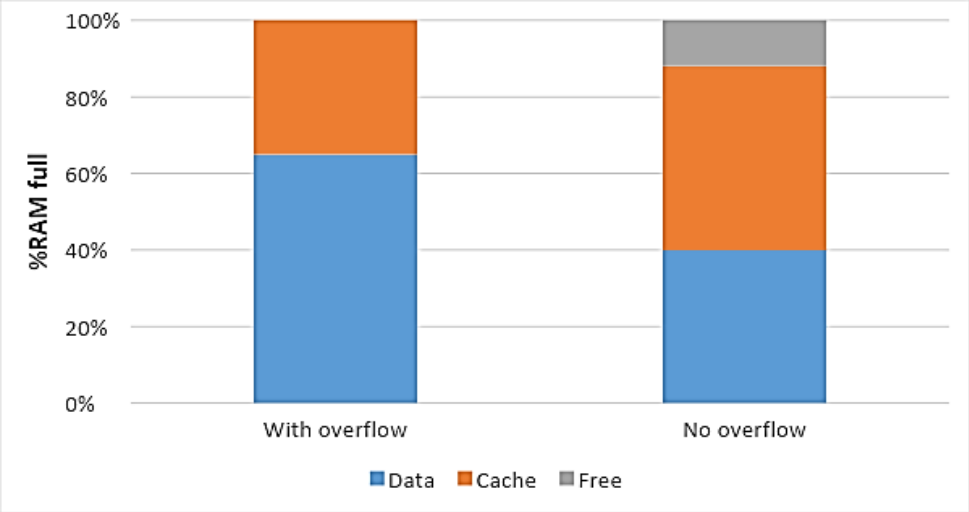
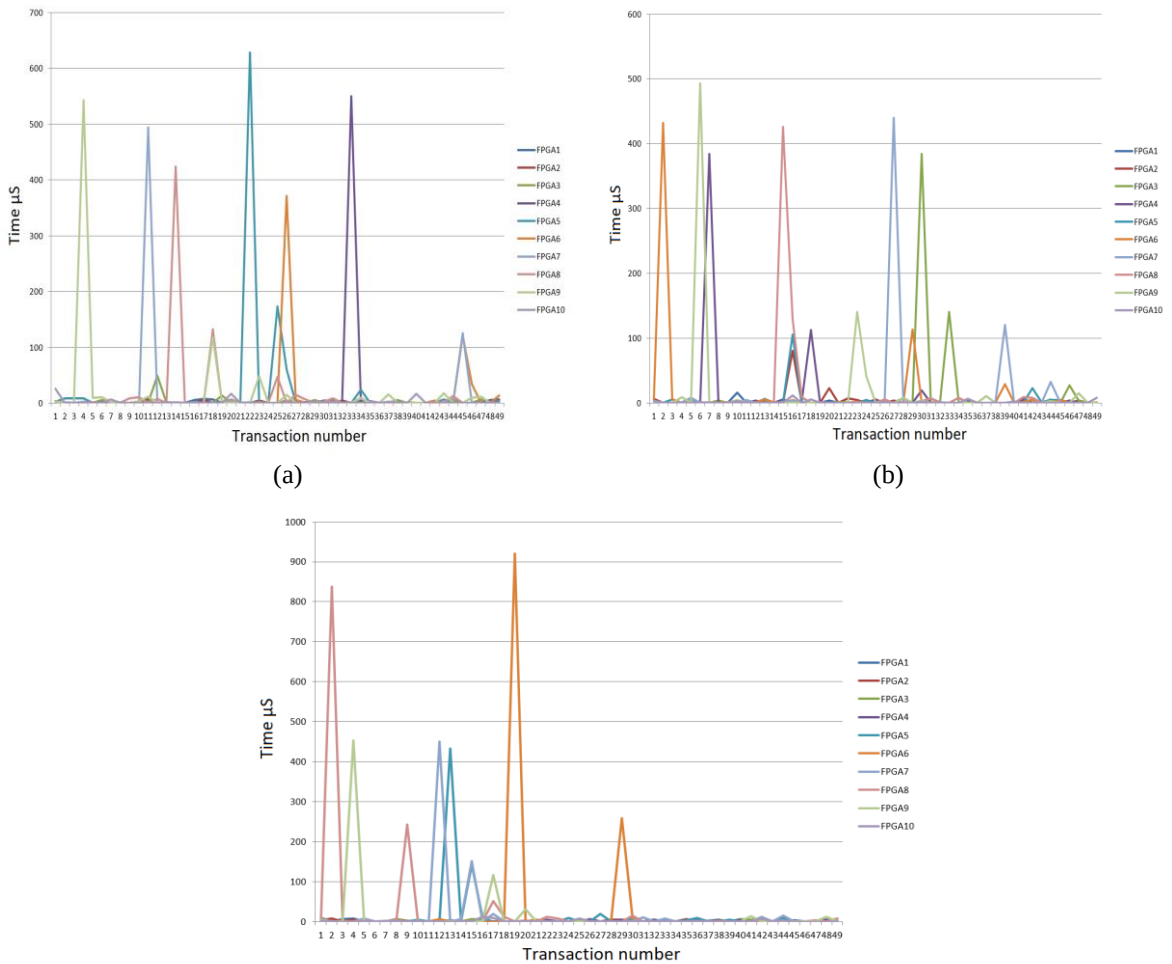


Figure 7. RAM distribution

Due to the fact that the implementation of caching on the FPGA is hardware, and the measurements are output via a serial port to another computer for further analysis. It becomes possible to investigate page faults of each of the computing nodes without reducing system performance. Page fault data was measured at each compute node at 1 MHz.

Consider the following option: a computational program is given in which 80% of the calculations occur in the memory allocated for a specific node and 20% of the data is requested from other nodes. Each computing node must perform 100,000 operations. In this case, in one case, 5% of the memory for the cache is taken from free space in RAM, and in the second, from the main memory available for data.

In Figure 8 we can see how much time it took each of the computing nodes to perform each operation at the busiest moment for the switching network, namely the beginning of calculations. It is at this moment that the largest flow of data occurs, both for caching and for obtaining the initial data pages. For example, without adding caching as shown in Figure 8(a), the duration of long page faults reached 630 microseconds, and the average time of long page faults is about 500 microseconds. After adding 5% cache from free memory as shown in Figure 8(b), long-time page faults decreased to an average of 400 microseconds. And after adding 5% cache from the data memory as shown in Figure 8(c), you can see that extremely long page faults appeared, but at the same time, both the number of long page faults and the duration, which in most cases did not reach 450 microseconds, decrease.



(a) (b)

(c)

Figure 8. Measuring page fault times: (a) without caching, (b) without overflow, and (c) overflow caching

Next, let's look at one of the computing nodes in more detail. As can be seen in Figure 9, in addition to long-term page faults, many short and medium-duration page faults occur during calculations. For example, if we consider a system without a cache as shown in Figure 9(a), the average duration of page faults is about 7 microseconds, and the page fault rate is about 40%. It is worth noting that such a high frequency of page faults occurs due to the fact that the computing node not only takes data pages from the system but also returns them. In the worst case, a situation may arise that the computing node will wait for the required page for a sufficiently long period of time, thereby forming a long-term page fault.

Figure 9(b) also shows that when adding cache from free memory, the average duration of page faults decreased from 7 to 6 microseconds, while reducing the number by an average of 3%. And after adding a cache from data memory as shown in Figure 9(c), the duration of page faults not only did not decrease, but in some places increased to 13 microseconds. It is worth noting that the number of page failures has significantly decreased. And it was about 20%.

Looking at a longer period of time as shown in Figure 10, one can notice an increase in the duration of page faults. This is due to the fact that an increasing number of pages are located in other computing nodes. Figure 11 shows the ratio of normal operations to page faults. As can be seen from these graphs, as pages move from one node to another, the number of page faults normalizes and, as a result, their duration stabilizes.

In Figures 12 and 13 show page fault durations before and after adding the cross-page. If we consider the stabilized period of calculations without caching as shown in Figure 12(a), then the average duration of line failures without a cache was about 15 microseconds, with peaks up to 30 microseconds. At the same time, with the addition of a cache from free memory as shown in Figure 12(b), the duration of page faults was reduced to 12 microseconds with peaks of up to 20-25 microseconds. And after adding a cache from data memory as shown in Figure 12(c), no significant changes were noticed.

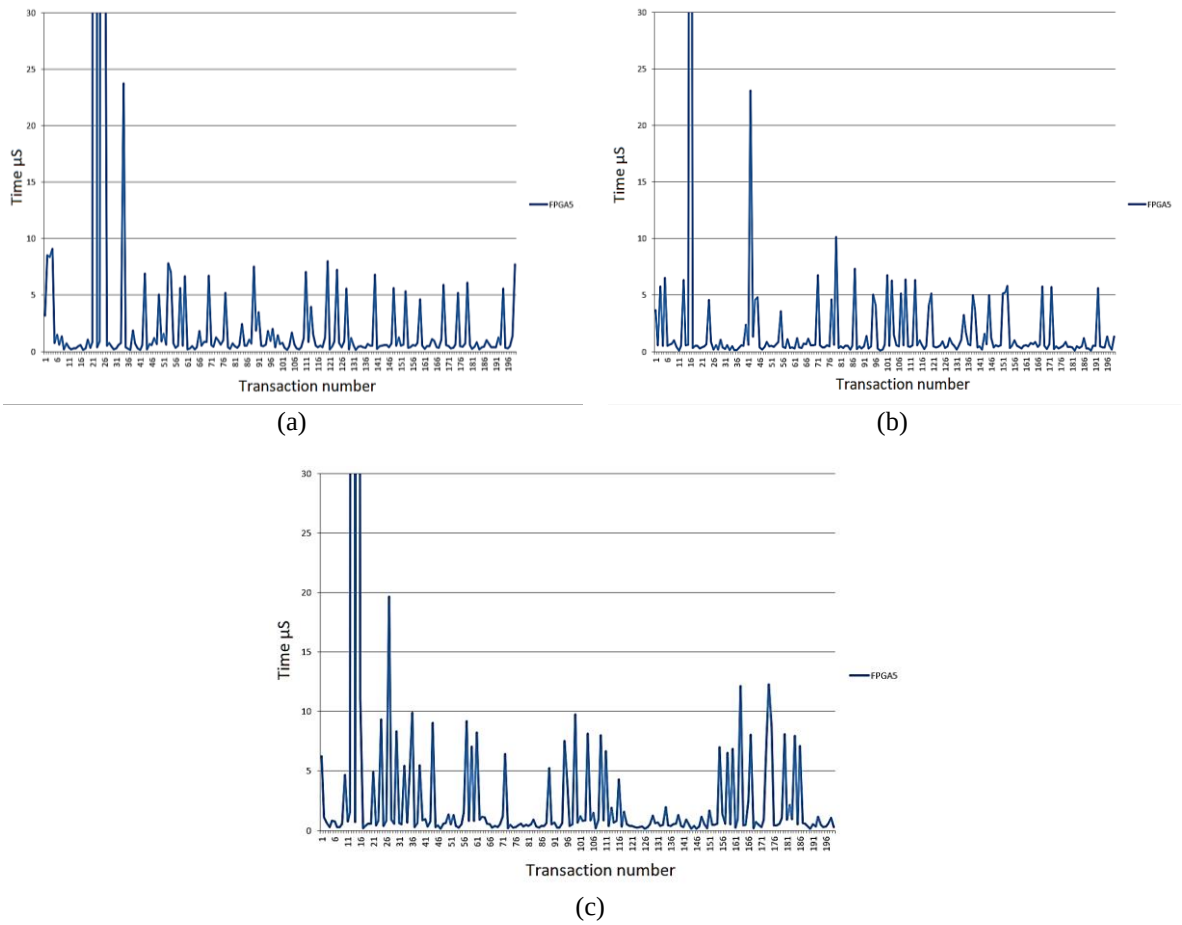


Figure 9. Measuring the time of page faults: (a) without caching, (b) with caching without overflow, and (c) with caching and overflow from 1 to 200 operations

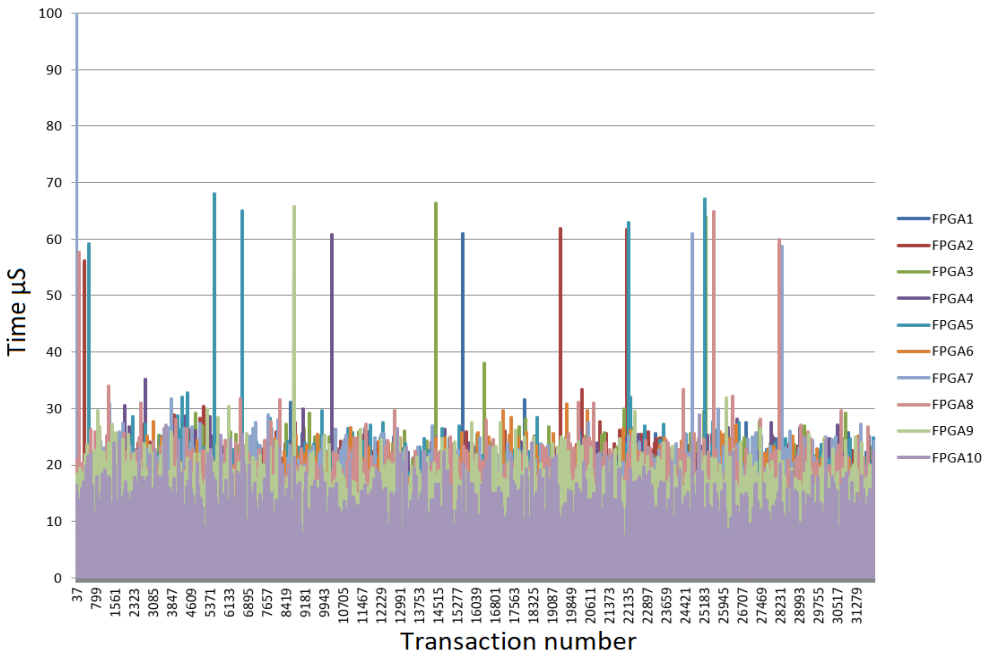


Figure 10. Measurement of page fault time without caching from 37 to 35,000 operations

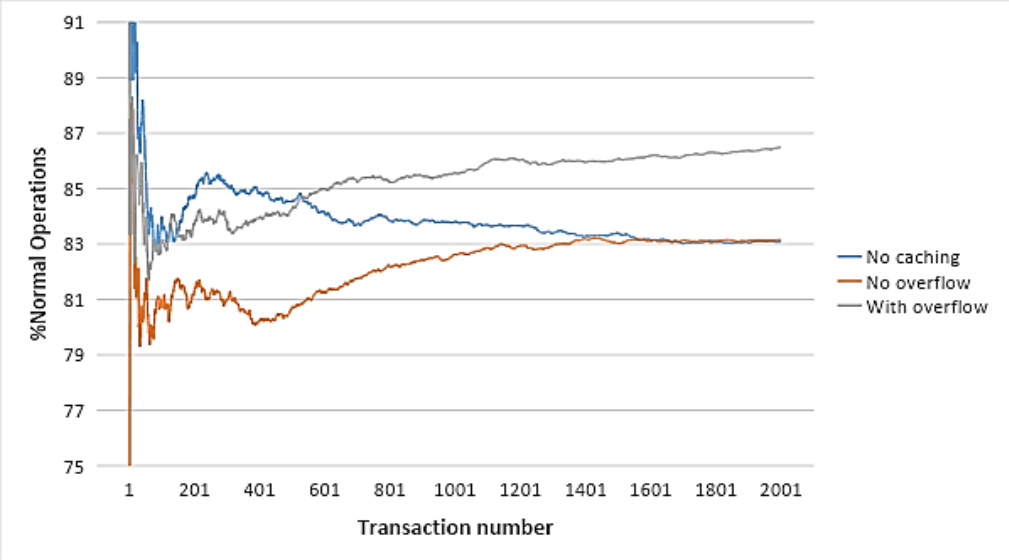


Figure 11. Percentage ratio of normal operations to page faults without caching from 1 to 2,000 operations

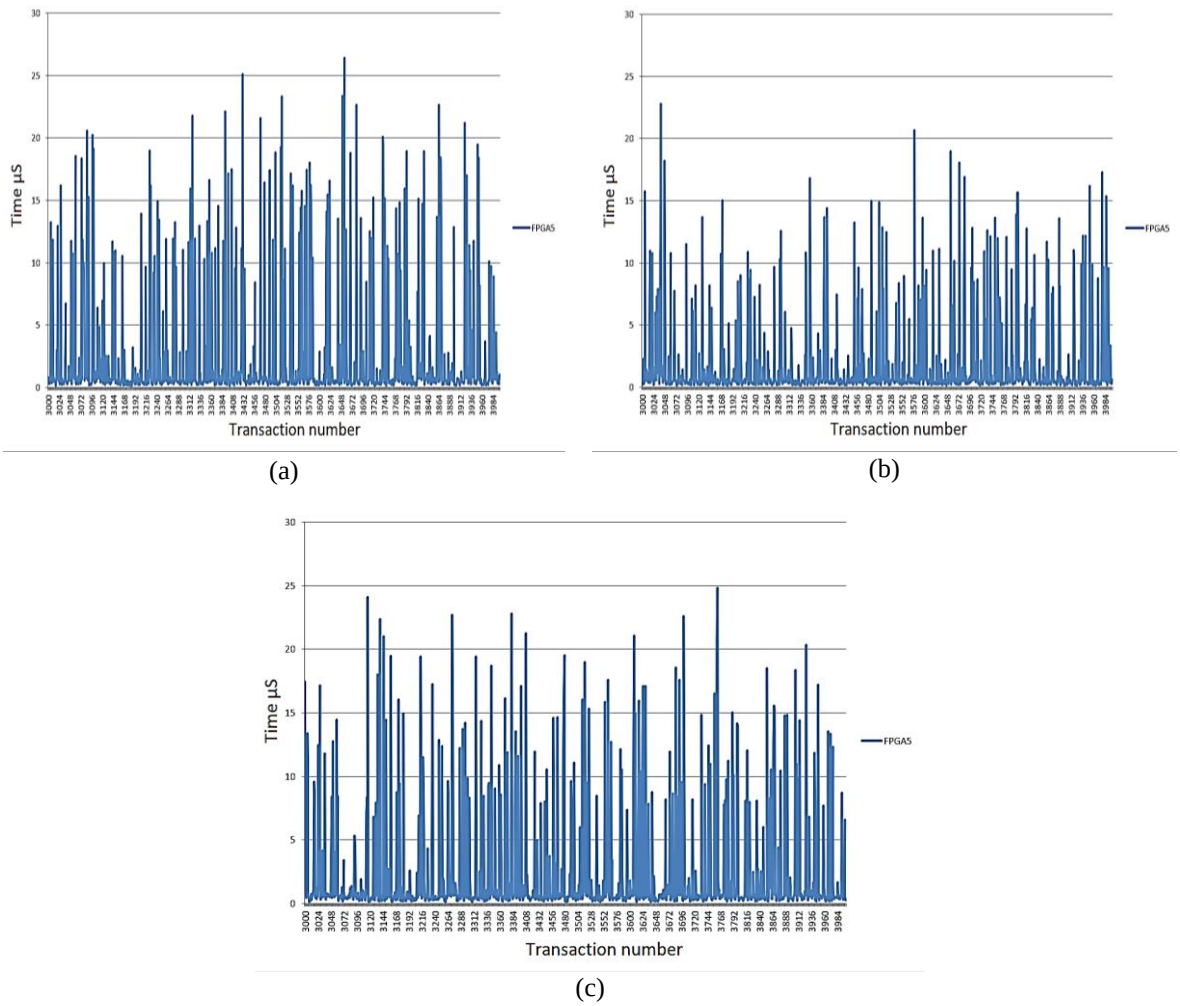
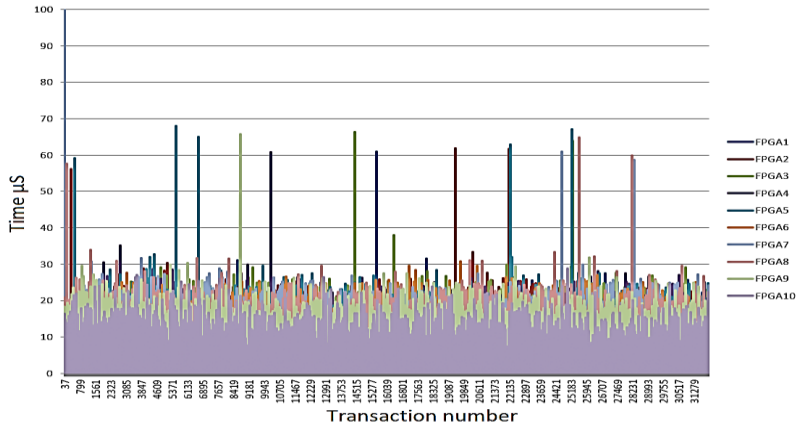
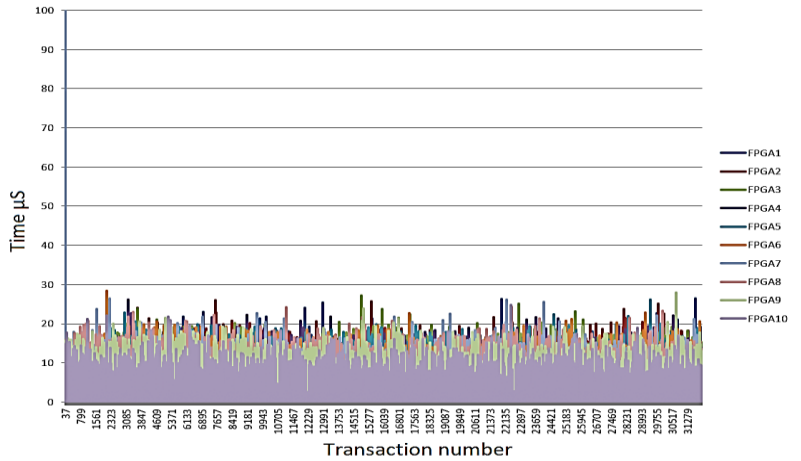


Figure 12. Measuring the time of page faults: (a) without caching, (b) with caching, and (c) with caching and overflow from 3,000 to 4,000 operations

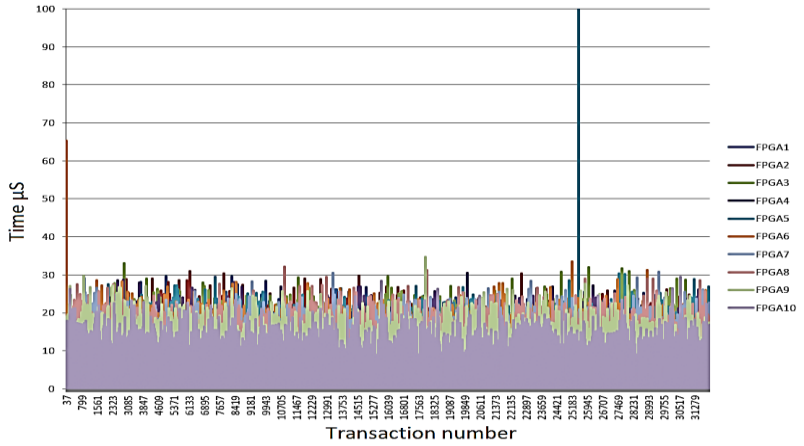
Measuring the time of page faults as shown in Figure 13. Moreover, if we consider a long period as shown in Figure 13(a) for very rare events, that is, long-term page faults. Adding a cache from free memory as shown in Figure 13(b) or adding a cache from data memory as shown in Figure 13(c) significantly reduces their number. Due to this, the performance of the system significantly increases.



(a)



(b)



(c)

Figure 13. Measuring the time of page faults: (a) without caching, (b) with caching without overflow, and (c) with caching and overflow from 37 to 35,000 operations

As a conclusion from the study on FPGA, it can be noted that the execution time without caching was 15 seconds. The execution time with a cache from free memory was 13 seconds, and the execution time with a cache from data memory was 14 seconds. Moreover, if you look at the program execution graph depending on the cache size at 20% of the requested data, you can clearly see the effectiveness of caching.

4. CONCLUSION

In the course of this work, the use of page caching in the RAM of computing nodes in multicomputer systems such as SVM using the working set strategy was investigated. An FPGA-based SVM system was proposed and implemented, and research was conducted to evaluate the effectiveness of caching in hardware implementation. In a study on FPGA, it was found that the execution time with a free cache was 13 seconds and the average duration of page faults decreased from 7 to 6 microseconds, while reducing the number by an average of 3%. The execution time with the cache from data memory was 14 seconds, but the average duration of page faults in some cases began to reach 13 microseconds and the number of page faults was reduced to 20%. While without caching this time is 15 seconds, the average page fault duration is about 7 microseconds, and the page fault rate is about 40%. The performance gain from adding a cache from free RAM memory with one of the worst memory access options was 14%. These results confirm the effectiveness of hardware-based caching. In general, this study can be useful for improving the performance of SVM systems under various application conditions and for developers of distributed computing systems.

ACKNOWLEDGEMENTS

This work was funded by Committee of Science of Republic of Kazakhstan AP09260767 "Development of intellectual information-analytical system for assessing the health of students in Kazakhstan" (2021-2023).

REFERENCES

[1] K. Torben and K. Andreas, "Efficient physical page migrations in shared virtual memory reconfigurable computing systems," *International Conference on Field-Programmable Technology*, December 2021, doi: 10.1109/ICFPT52863.2021.9609831.
[2] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, January 2019, pp. 48-60, doi: 10.1145/3282307.
[3] L. Oden and T. Saidi, "Implementation and evaluation of CUDA-unified memory in numba," *Euro-Par 2020: Parallel Processing Workshops*, March 2021, pp. 197-208, doi: 10.1007/978-3-030-71593-9_16.
[4] M. K. Y. Ngetich, C. Otieno, and M. Kimwele, "A model for code restructuring, a tool for improving systems quality in compliance with object oriented coding practice," *IJCSI International Journal of Computer Science Issues*, vol. 16, no. 3, May 2019, pp. 32-36, doi: 10.5281/zenodo.3252971.
[5] C. D. Newman, B. Bartman, M. L. Collard, and J. I. Maletic, "Simplifying the construction of source code transformations via automatic syntactic restructurings," *Journal of Software: Evolution and Process*, vol. 29, no. 4, January 2017, doi: 10.1002/smr.1831.
[6] M. Sinha and S. Jain, "A survey paper on program restructuring and component reuse with data mining technique," *International Journal of Engineering Research and General Science*, vol. 4, no. 3, May 2016, pp. 578-581.
[8] C. Lung, X. Xu, M. Zaman, and A. Srinivasan, "Program restructuring using clustering techniques," *Journal of Systems and Software*, vol. 79, no. 9, September 2006, pp. 1261-1279, doi: 10.1016/j.jss.2006.02.037.
[9] Z. Marian, I. Czibula, and G. Czibula, "A hierarchical clustering-based approach for software restructuring at the package level," *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, September 2017, pp. 239-246, doi: 10.1109/SYNASC.2017.00046.
[10] D. Ferrari, *Computer Systems Performance Evaluation (1st Edition, Prentice Hall)*, April 1978.
[11] J. Paulo, L. Machado, E. V. P. Paula-Sobrinho, and M. A. Maia, "Anti-bloater class restructuring: An exploratory study," *Journal of Software: Evolution and process*, vol. 34, no. 3, February 2022, doi: 10.1002/smr.2431.
[12] S. Jaiswal and M. S. Verma, "A survey on software component restructuring," *International Journal of Advance Research and Development*, vol. 2, no. 5, May 2017, pp. 38-41.
[13] Y. A. Divya, "An efficient virtual memory using graceful code," *International Journal of Trend in Scientific Research and Development*, vol. 3, no. 4, June 2019, pp. 623-626, doi: 10.31142/ijtsrd23878.
[14] D. Steinhöfel, "Ever change a running system: structured software reengineering using automatically proven-correct transformation rules," *Ernst Denert Award for Software Engineering 2020. Springer, Cham*, February 2022, pp. 197-226, doi: 10.1007/978-3-030-83128-8_10.
[15] R. Hähnle, A. T. Heydari, A. Mazaheri, M. Norouzi, D. Steinhöfel, and F. Wolf, "Safer parallelization," *Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles*, LNTCS, vol. 12477, October 2020, pp. 117-137, doi: 10.1007/978-3-030-61470-6_8.
[16] S. Kaur, A. Kaur, and G. Dhiman, "Deep analysis of quality of primary studies on assessing the impact of refactoring on software quality," *Materials Today: Proceedings*, January 2021, doi: 10.1016/j.matpr.2020.11.217.
[17] S. Kaur and P. Singh, "How does object-oriented code refactoring influence software quality? research landscapes and challenges," *Journal of Systems and Software*, vol. 157, November 2019, doi: 10.1016/j.jss.2019.110394.
[18] G. Bavota, A. D. Lucia, and R. Oliveto, "Identifying extract class refactoring opportunities using structural and semantic cohesion measures," *Journal of Systems and Software*, vol. 84, no. 3, March 2011, pp. 397-414, doi: 10.1016/j.jss.2010.11.918.

[19] M. Gatrell and S. Counsell, "The effect of refactoring on change and fault-proneness in commercial C# software," *Science of Computer Programming*, vol. 102, March 2015, pp. 44-56, doi: 10.1016/j.scico.2014.12.002.

[20] S. Singh and S. Kaur, "A systematic literature review: Refactoring for disclosing code smells in object oriented software," *Ain Shams Engineering Journal*, vol. 9, no. 4, December 2018, pp. 2129-2151, doi: 10.1016/j.asej.2017.03.002.

[21] J. A. Dallal, "Identifying refactoring opportunities in object-oriented code: a systematic literature review," *Information and Software Technology*, vol. 58, February 2015, pp. 231-249, doi: 10.1016/j.infsof.2014.08.002.

[22] M. Misbhauddin and M. Alshayeb, "UML model refactoring: a systematic literature review," *Empirical Software Engineering*, vol. 20, February 2015, pp. 206-251, doi: 10.1007/s10664-013-9283-7.

[23] M. Fowler, *Refactoring: Improving the Design of Existing Code (2nd Edition)*, November 2018.

[24] S. Kaur and S. Singh, "Object oriented metrics based empirical model for predicting "code smells" in open source software," *Journal of The Institution of Engineers*, vol. 104, January 2023, pp. 241-257, doi: 10.1007/s40031-022-00833-4.

[25] D. Tenorio, A. C. Bibiano, and A. Garcia, "On the customization of batch refactoring," *IEEE/ACM 3rd International Workshop on Refactoring*, September 2019, doi: 10.1109/IWoR.2019.00010.

[26] Q. Gu, Y. Chang, X. Li, Z. Chang, and Z. Feng, "A novel F-SVM based on FOA for improving SVM performance," *Expert Systems with Applications*, vol. 165, March 2021, doi:10.1016/ j.eswa.2020.113713.

[27] M. Cai, D. Zhang, H. Huang, and A. Scalable, "Virtual memory system based on decentralization for many-cores," *Journal of Systems Architecture*, vol. 107, August 2020, doi:10.1016/ j.sysarc.2020.101803.

[28] Yu. I. Zhuravlev, "Algebras over sets of incorrect (heuristic) algorithms. II," *Kibernetika*, no. 4, November 1977, pp. 37–45.

[29] Yu. I. Zhuravlev, "Algebras over sets of incorrect (heuristic) algorithms.III," *Kibernetika*, no. 2, March 1978, pp. 35–43.

[30] Yu. I. Zhuravlev, "Principles of construction of justification of algorithms for the solution of badly formalized problems," *Math. Notes*, vol. 23, pp. 493-501, June 1978.

[31] Yu. I. Zhuravlev, "The way to generate recognition algorithms being correct for the given reference sample," *Mat. Mat. Fiz*., vol. 19, no. 3, pp. 200–208, 1979.

[32] A. E. Dyusembaev, "Correct models of program segmenting. Journal of pattern recognition and image," *Analises USA*, vol. 3, no. 6, 1993, pp. 187-204.

[33] A. E. Dyusembaev, "Mathematical models of program segmentation," *M: Fizmatlit (Nau- ka, MAIK)*, 2001.

[34] L. R. Foulds, "Combinatorial optimization," *–Berlin-Heidelberg -New York-Tokyo: Springer Verlag*, 1984.

[35] T. Masuda, H. Shiota, K. Noguchi, and T. Ohki, "Optimization of program organization by cluster analysis," *Proceeding of IFIP Congress*, 1974, pp. 261-266.

[36] S. Kambampati, J. S. Gray, and H. A. Kim, "Level set topology optimization of structures under stress and temperature constraints," *Computers and Structures*, vol. 235, July 2020, doi: 10.1016/j.compstruc.2020.106265.

[37] R. Nie, B. He, D.H. Hodges, and X. Ma, "Form finding and design optimization of cable network structures with flexible frames," *Computers and Structures*, vol. 220, August 2019, pp. 81-91, doi: 10.1016/j.compstruc.2019.05.004.

[38] M. Jahangiri, M. A. Hadianfard, M. A. Najafgholipour, M. Jahangiri, and M. R. Gerami, "Interactive autodidactic school: A new metaheuristic optimization algorithm for solving mathematical and structural design optimization problems," *Computers and Structures*, vol. 235, July 2020, doi: 10.1016/j.compstruc.2020.106268.

[39] J. Zhang, M. Xiao, P. Li, and L. Gao, "Sampling-based system reliability-based design optimization using composite active learning Kriging," *Computers and Structures*, vol. 239, October 2020, doi: 10.1016/j.compstruc.2020.106321.

[40] V. R. Kommareddy, J. Kotra, C. Hughes, S. D. Hammond, and A. Avad, "PreFAM : Understanding the impact of prefetching in fabric-attached memory architectures," *MEMSYS 2020: The International Symposium on Memory Systems*, September 2020, pp. 323–334, doi: 10.1145/3422575.3422804.

[41] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, "Kleio: A hybrid memory page scheduler with machine intelligence," *HPDC '19: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, June 2019, pp. 37–48, doi: 10.1145/3307681.3325398.

[42] J. H. Ryoo, M. R. Meswani, A. Prodromou, and L. K. John, "SILC-FM: Subblocked interleaved cache-like flat memory organization," *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, February 2017, doi: 10.1109/HPCA.2017.20.

[43] M. Poremba, I. Akgun, J. Yin, O. Kayiran, Y. Xie, and G. H. Loh, "There and back again: optimizing the interconnect in networks of memory cubes," *ISCA '17: Proceedings of the 44th Annual International Symposium on Computer Architecture*, June 2017, pp. 678–690, doi: 10.1145/3079856.3080251.

[44] P. J. Denning, "Working set analytics," *ACM Computing Surveys*, vol. 53, no. 6, art. 113, February 2021, doi: 10.1145/3399709.

[45] S. Vyazigin and M. Mansurova, "Combinatorial aspect of code restructuring for virtual memory computer systems under WS swapping strategy," *International Conference on Parallel Computing Technologies. PaCT 2023: Parallel Computing Technologies*, August 2023, pp 136–147, doi: 10.1007/978-3-031-41673-6_11.

[46] S. Vyazigin, A. Dyusembaev, and M. Mansurova, "Emulation of x86 computer on FPGA," *2020 IEEE 8th Workshop on Advances in Information, Electronic and Electrical Engineering*, May 2021, doi: 10.1109/AIEEE51419.2021.9435812.

[47] S. D. Kumar and M. Roopa, "Design and analysis of multiple read port techniques using bank division with XOR method for multi-ported-memory on FPGA platform," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 6, pp. 4785-4793, December 2021, doi: 10.11591/ijece.v11i6.pp4785-4793.

[48] T. L. Prasanna, N. Siddaiah, B. M. Krishna, M. R. Valluri, "Implementation of the advanced encryption standard algorithm on an FPGA for image processing through the universal asynchronous receiver-transmitter protocol," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 6, pp. 6114-6122, December 2022, doi: 10.11591/ijece.v12i6.pp6114-6122.

[49] S. Che, M. Orr, and J. Gallmeier, "Work stealing in a shared virtual-memory heterogeneous environment: A case study with betweenness centrality," *CF'17: Proceedings of the Computing Frontiers Conference*, May 2017, pp. 164–173, doi: 10.1145/3075564.3075567.

[50] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and leveraging shared L1 caches in GPUs," *PACT '20: Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, September 2020, pp. 161–173, doi: 10.1145/3410463.3414623.

BIOGRAPHIES OF AUTHORS

Stepan Vyazigin obtained his undergraduate Almaty Technological University and obtained his graduate degree from Al-Farabi Kazakh National University. Currently, he works as engineer at the LLP “Special Design Technical Bureau "Granit". His research area include computer systems engineering, parallel computing and artificial intelligence fields. He can be contacted at email: wismas1996@gmail.com.

Madina Mansurova is Candidate of Physical and Mathematical Sciences, Associate Professor, Head of the Department of Artificial Intelligence and Big Data, KazNU named after al-Farabi, leading researcher at the Research Institute of Mathematics and Mechanics, KazNU named after Al-Farabi, an expert in high performance computing and intelligent data processing. Mansurova M.E. in 1994, she graduated with honors from the Faculty of Mechanics and Mathematics of the Kazakh State University Al-Farabi with a degree in Applied Mathematics. Since 2001 he has been working at KazNU named after al-Farabi. In 2007 she defended her thesis on the topic "Construction of the reachability set of controlled systems" in specialty 01.01.10 - Mathematical theory of controlled systems, she was awarded the degree of candidate of physical and mathematical sciences. In 2011, she was awarded the academic title of Associate Professor in the specialty "Informatics, Computer Engineering and Management". Mansurova M.E. - the author of more than 90 scientific articles, 3 monographs, 2 textbooks with the stamp of the Ministry of Education and Science of the Republic of Kazakhstan, over 10 teaching aids. She can be contacted at email: madina.mansurova@kaznu.kz.

Victor Malyshkin is Doctor of Technical Sciences, Professor, Head of the department of Parallel Program Synthesis Laboratory, Institute of Computational Mathematics and Mathematical Geophysics SB RAS. Expert in the field of parallel programming technologies and design of parallel programs. Malyshkin V.E. graduated from Tomsk State University in 1970 with a degree in Mathematics, theory of functions. In 1984 he defended his thesis on the topic "Parallel Program Synthesis on the Basis of Computational Models" in specialty 01.01.00 - Mathematics, he was awarded the academic degree of Candidate of Physical and Mathematical Sciences. In 1993, he defended his dissertation on the topic “Organization of Parallel Computations on Large-Block Multicomputers,” specialty 05.13.16 - Application of computer technology, mathematical modeling and mathematical methods in scientific research, he was awarded the academic degree of Doctor of Technical Sciences. Member of the editorial board of journals The International Journal of Computational Science and Engineering, The Journal of Supercomputing, The International Journal of Big Data Intelligence (IJDBI), InderScience Publisher and some other journals. Malyshkin V.E. - the author of more than 100 scientific articles. He can be contacted at email: malysh@ssd.sscc.ru.

Aygul Shaykhulova is PhD in Engineering, Associate Professor at the Department of Mechanical Engineering Technology. Specializes in automation and digitization of production, technology transfer. She participated in projects related to the reconstruction and technical re-equipment of leading defense industry enterprises. She contributed to targeted programs for developing roadmaps for the digitization of production. She manages issues related to digital technologies and R&D transfer. Author of 30 publications, including 2 monographs and 4 patent certificates. She can be contacted at email: shaihulova@inbox.ru.