ISSN 2079-3316 **PROGRAM SYSTEMS: THEORY AND APPLICATIONS** **vol. 10**, No 1(40), pp. 19–32

UDC 004.942

E. A. Barkovsky, A. A. Lazutina, A. V. Sokolov

The Optimal Control of Two Work-Stealing Deques, Moving One After Another in a Shared Memory

Abstract. In the parallel work-stealing load balancers, each core owns personal buffer of tasks called deque. One end of the deque is used by its owner to add and retrieve tasks, while the second end is used by other cores to steal tasks. In the paper two representation methods of deques are analyzed: partitioned serial cyclic representation of deques (one of the conventional techniques); and the new approach proposed by our team, without partition of shared memory in advance between deques moving one after another in a circle. Previously we analyzed these methods for representing FIFO queues in network applications, where the "One after another" way gave the best result for some values of the system parameters.

Purpose of this research is to construct and analyze models of the process of work with two circular deques located in shared memory, where they movie one after another in a circle. The mathematical model is constructed in the form of a random walk by integer points in the pyramid. The simulation model is constructed using the Monte Carlo method. The used work-stealing strategy is stealing of one element. We propose the mathematical and simulation models of this process and carry out numerical experiments.

Key words and phrases: Work-Stealing Schedulers, Work-Stealing Deques, Data Structures, Absorbing Markov Chains, Random Walks.

2010 *Mathematics Subject Classification*: 68Q85; 68P05, 68Q87

Introduction

There are two main strategies of parallel computations balancing: static and dynamic. In the static strategies, it is assumed that the order of tasks execution is known in advance. In that case, it is possible to

This work was supported by grant RFBR No 18-01-00125-a.

10.25209/2079-3316-2019-10-1-19-32

construct the optimal schedule before the work started. This situation is rarely achievable and the solution, in this case, is an NP-complete problem.

The dynamic strategies use simplified schemes: *work-sharing*—tasks are transmitted from a more loaded core to a less loaded one; *work-stealing*—empty cores steal tasks from other cores [1, 2].

Work-stealing is used in such systems as Cilk [3], Cilk++, TPL [4], X10 [5], TBB, JSR166 (java.util.concurrent package), Erlang, OpenMP in ICC [6]. Each core executes tasks, pointers to which are stored in its deque. If a new task is created, the core adds its pointer to the deque; if core needs a task, it reads a pointer from the top of the deque. But if it finds that the deque is empty, the core will start to steal pointers from deques of the other cores. Stealing occurs from the bottom of the deque—similar to FIFO-queues—while operations of insertion and deletion are executed in the LIFO order. In the book by D. Knuth, this data structure is called "an input restricted deque" [7]. Some of the work-stealing strategies are: stealing of one element [1], stealing of half the elements [8].

There are several ways to represent the input restricted deques in memory, for example, using linked lists. In [9] the models of linked representation of stacks and queues are described. The model of linked representation of deques can be built in the same way.

For stacks and queues, the paged implementation can be used in the form of a single-linked list with pages of the same length. This method was presented and analyzed in [10]. In [11] a variant of this method was proposed for deques, only it requires a double-linked list. The model of work-stealing balancer built on the basis of the queuing theory was described in [12], but no specific ways of representing deques in memory were considered.

In this paper, we analyze the work of two deques. While this particular case is the beginning of the research, such model can be already used in practice, for example, in the architectures where the cache memory is absent. Thus, in SEAforth architecture each core has two stacks (for storing data and return addresses), and there are two FIFO-queues per core in AsAP-II architecture [13]. In these architectures stacks and queues are implemented cyclically and separated from each other, and the overflow of data structures may cause loss of elements. We assume that it is possible to implement work-stealing deques in a similar way and build desired chips by composing them from a "two-deques" ones.

In this paper, we propose simulation and mathematical models of the new method of representation of work-stealing deques. They move one after

another in a circle in the same partition of shared memory (the method is patented [14]). For FIFO-queues, a similar method was proposed in [15] and analyzed in [16]. It is possible to implement deques in RAM or in registers, so here we do not specify the type of memory.

The mathematical model of this process is built as a random walk on integer points in a pyramid. The transitions are carried out by the discrete process with given probabilities. Preliminary results were reported in [19]. Previously, such models were built by our team for some other dynamic data structures: FIFO-queues, stacks, priority queues [9, 10, 15, 17, 18].

Our team proposed and analyzed models, where sequential deques are located in the separated parts of shared memory [20], with the discrete execution of operations [21–23]. On the basis of the models, optimization problems were solved. The optimality criterion is the maximum mathematical expectation of time before the redistribution (overflow) of memory.

Such an optimality criterion is useful in applications, where memory overflow is an emergency. For example, real-time applications of work-stealing load balancers or hardware implementations of deques [24].

We also note that the question of the correctness of using a probabilistic approach to model the behavior of data structures in parallel programs is a legitimate one. First of all, it should be noted that even in classical sequential programs the order of operations with data structures depends on the input data and is determined in real time. In parallel programs, non-determinism is further enhanced. Here you can quote from the work [25]: "Parallel programming fundamentally cannot completely get rid of non-determinism, since the corresponding programming tools—processes, threads, and their interaction through a common resource—are required for effective implementations on modern hardware, and also because of the distributed nature of applications functioning in the real world".

The probabilities of operations on deques (sequential cyclic representation) were estimated in [26] using statistic data collected from several tests in [27]. The obtained probabilities of operations were used in numerical experiments to analyze the developed models. Statistics were collected using a version of the load balancer, where deques store pointers to the tasks. In this version, it is necessary to refer to the memory allocator twice for each task: allocate memory of the task object and then release it when the task is completed. Thus, the associated with the memory allocator overhead arises.

The memory manager implemented [28] in the aforementioned version of the work-stealing load balancer. In this manager, the following memory management methods based on the developed models [26] were implemented and analyzed: optimal partition, partition in half. The study showed that when shared memory is partitioned optimally, the tasks require less memory, but the mathematical expectation of the time of their solution becomes longer. Such characteristics of the manager can be useful in the real-time applications, where a small increase in operating time may be acceptable, while memory overflow leads to the program crash.

The new version of the load balancer is described in [29]. There, objects are stored in deques, which are represented in the heap as doubly-linked lists of arrays. To work with deques, one can use the same models and methods as in the previous version. The difference is, in the new version deques take up more memory, because task objects are larger than pointers to tasks.

Usually, in the work-stealing load balancers work with deques happens via cyclic arrays. For the deques, arrays are allocated from the heap using classic memory functions in C/C++ translators. When deques grow or shrink significantly, new arrays of larger or smaller size are allocated.

In this paper we solve the problem of finding the optimal memory allocation algorithms for deques, assuming that the probabilities of operations performed on them are known.

1. The Mathematical Model

Suppose that the size of shared memory is m units. Two cyclic work-stealing deques moving one after another in a circle are located in this shared memory. Empty deque continues its work/movement from the middle of the vacant memory. The movement pattern of the data structures is shown in Figure 1.

The entire work of the system is divided into operations, where elements are inserted and/or deleted per unit of time.

The probabilities of operations are the following:

- With the probability p_1 insertion in the I deque occurs;
- With the probability p_2 insertion in the II deque occurs;
- With the probability p_{12} parallel insertion in both deques occurs;
- With the probability q_1 deletion from the I deque occurs;
- With the probability q_2 deletion from the II deque occurs;

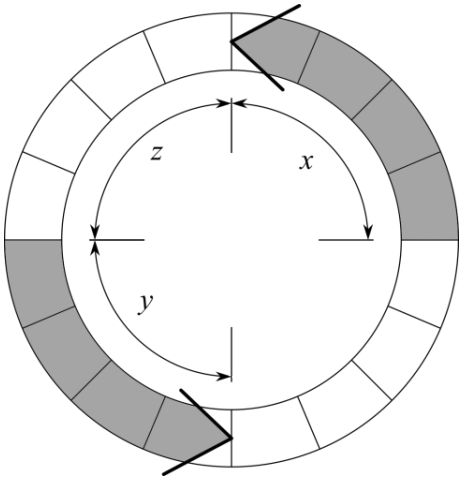

Figure 1. The movement scheme of two work-stealing deques moving one after another in a circle

- With the probability q_{12} parallel deletion from both deques occurs;
- With the probability pq_{12} insertion in the I deque and deletion from the II one occur;
- With the probability pq_{21} insertion in the II deque and deletion from the I one occur;
- With the probability r size of deques does not change (for example, reading operation).

$$p_1 + p_2 + p_{12} + q_1 + q_2 + q_{12} + pq_{12} + pq_{21} + r = 1.$$

If the system is trying to delete an element from the empty deque, then the work-stealing process starts: empty deque steals work (an element) from the other deque. The following work-stealing strategy was used: stealing of one element.

The task is to determine the mean operating time of the system to memory overflow, and compare it with the mean operating time of the system based on the cyclical organization of deques.

Denote the current lengths of the first and the second deques as x and y, and the distance between them as z. The mathematical model of the process is a random walk inside the integer pyramid, with the top $(0, 0, 0)$ and the base $x + y + z = m$ (Figure 2).

Schemes of transitions between states are shown below. Here (x, y, z)

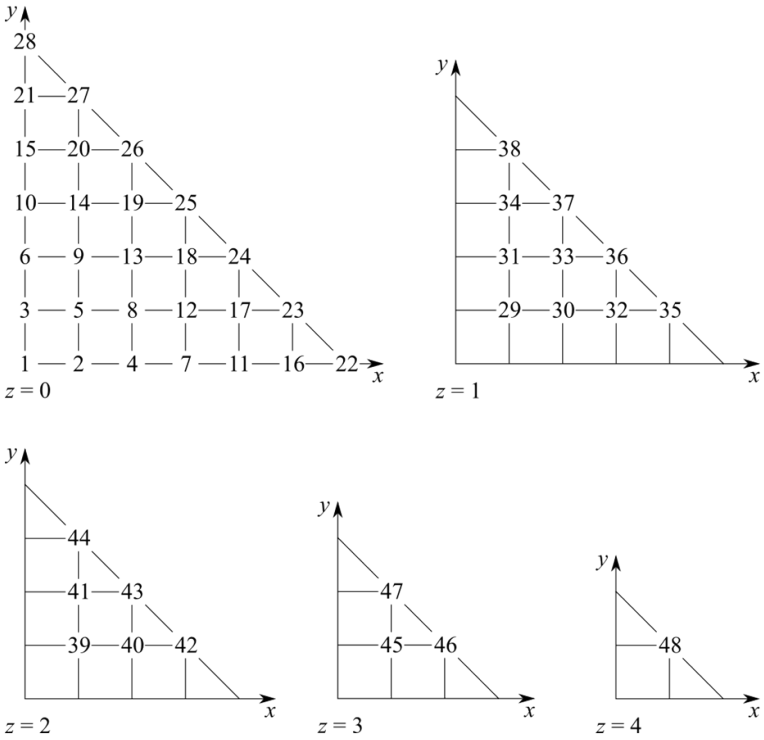

FIGURE 2. The area of walk and the numbering of states for two deques moving one after another in the memory of size $m = 6$

is the previous state of the process, and (x', y', z') is the new state of the process.

The walk inside the pyramid is as follows:

$$
\begin{aligned}
&(x, y, z) \xrightarrow{r} (x, y, z)\\
&(x, y, z) \xrightarrow{p_1} (x', y', z')\\
&\qquad = \begin{cases} (x+1, y, z-1), & x, y, z > 0,\ x+y+z < m \\ (x+1, y, z), & 0 < x < m,\ y = 0,\ z = 0 \\ (x+1, y, z + [\frac{m-y-1}{2}]), & x = 0,\ 0 < y < m,\ z = 0 \end{cases}\\
&(x, y, z) \xrightarrow{p_2} (x', y', z')
\end{aligned}
$$

$$= \begin{cases} (x, y+1, z), & x, y, z > 0,\ x+y+z < m \\ (x, y+1, z+[\frac{m-x-1}{2}]), & 0 < x < m,\ y = 0,\ z = 0 \end{cases}$$

$(x, y, z) \xrightarrow{p_{12}} (x', y', z')$

$$= \begin{cases} (x+1, y+1, z-1), & x, y, z > 0,\ x+y+z < m \\ (x+1, y+1, z+[\frac{m-x-2}{2}]), & 0 < x < m-1,\ y = 0,\ z = 0 \\ (x+1, y+1, z+[\frac{m-y-2}{2}]), & x = 0,\ 0 \le y < m-1,\ z = 0 \end{cases}$$

$(x, y, z) \xrightarrow{q_1} (x', y', z')$

$$= \begin{cases} (x-1, y, z+1), & x, y, z > 0,\ x+y+z \le m \\ (x+1, y-1, z+[\frac{m-y}{2}]), & x = 0,\ 1 < y \le m,\ z = 0 \\ (x, y, z), & x = 0,\ 0 \le y \le 1,\ z = 0 \end{cases}$$

$(x, y, z) \xrightarrow{q_2} (x', y', z')$

$$= \begin{cases} (x, y-1, z), & x, y, z > 0,\ x+y+z \le m \\ (x-1, y+1, z+[\frac{m-x}{2}]), & 1 < x \le m,\ y = 0,\ z = 0 \\ (x, y, z), & 0 \le x \le 1,\ y = 0,\ z = 0 \end{cases}$$

$(x, y, z) \xrightarrow{q_{12}} (x', y', z')$

$$= \begin{cases} (x-1, y-1, z+1), & x, y, z > 0,\ x+y+z \le m \\ (x+1, y-2, z+[\frac{m-y}{2}]), & x = 0,\ 2 < y \le m,\ z = 0 \\ (x-2, y+1, z+[\frac{m-x+1}{2}]), & 2 \le x \le m,\ y = 0,\ z = 0 \\ (x-1, y, z), & 0 < x \le 2,\ y = 0,\ z = 0 \\ (x+1, y-2, z), & x = 0,\ y = 2,\ z = 0 \\ (x, y-1, z), & x = 0,\ y = 1,\ z = 0 \\ (x, y, z), & x = 0,\ y = 0,\ z = 0 \end{cases}$$

$(x, y, z) \xrightarrow{pq_{12}} (x', y', z')$

$$= \begin{cases} (x+1, y-1, z-1), & x, y, z > 0,\ x+y+z \le m \\ (x, y+1, z+[\frac{m-x-1}{2}]), & 0 < x < m,\ y = 0,\ z = 0 \\ (x+1, y-1, [\frac{m-y}{2}]), & x = 0,\ 0 < y < m,\ z = 0 \\ (x+1, y, z), & x = 0,\ y = 0,\ z = 0 \end{cases}$$

$(x, y, z) \xrightarrow{pq_{21}} (x', y', z')$

$$= \begin{cases} (x-1, y+1, z+1), & x, y, z > 0,\ x+y+z \le m \\ (x-1, y+1, z+[\frac{m-x-1}{2}]), & 0 < x < m,\ y = 0,\ z = 0 \\ (x+1, y, z+[\frac{m-y-1}{2}]), & x = 0,\ 0 < y < m,\ z = 0 \\ (x, y+1, z), & x = 0,\ y = 0,\ z = 0 \end{cases}$$

Then, based on the numbering of states and transition schemes, the

matrix of transition probabilities P is constructed. This matrix has a submatrix Q, which describes the process before it leaves the set of non-return states. This submatrix is needed to compose the fundamental matrix N of the absorbing chain $N = (I - Q)^{-1}$, where I is the identity matrix. The sum of the elements of the N matrix in the line corresponding to the initial state is the mean time to absorption (overflow) if the process started from zero ($x = 0$ and $y = 0$) [30].

2. The simulation model

To confirm the results of mathematical modeling, simulation modeling was used (Monte Carlo methods). Parameters of the simulation model are sizes of deques (x, y), the distance between the tail of the first deque and the head of the second one (z) and probabilities of operations (p_1, p_2, p_{12}, q_1, q_2, q_{12}, pq_{12}, pq_{21}, r).

To determine which operation to execute, an interval from 0 to 1 must be divided according to the probabilities. Next, the random number generator (standard gcc generator was used) generates a sequence of operations. Deques move one after another in a circle according to this sequence (Figure 1) while $z \neq -1$ or $x + y + z \neq m + 1$. The result of the model is the number of steps to memory overflow.

3. Some Examples of Numerical Analysis

To analyze the described in the paper "One after another" method of representation of work-stealing deques, it needs to be compared with the already analyzed method, namely the sequential cyclic method, where the shared memory is divided in half [26]. To do this, a series of experiments were conducted based on several sets of input data.

The results of some calculations with mathematical models are given in Table 1, Table 2, and Table 3 (these results were confirmed by simulation models). The analytical solution for this problem was not obtained, thus calculations must be performed for the specific memory sizes (value m). Sizes $m = 4, 6, 8, 10, 100$ are given as an example.

In Table 1 input data are the theoretical case of equal probabilities. In Table 2 and Table 3 probability estimates (in frequency) are taken. They occur when the system is engaged in solving the following problems: matrix multiplication, knapsack problem. The frequencies were obtained as a result of experiments with the work-stealing load balancer [27]. For these input data, the additional calculations were carried out with simulation models, where the memory size $m = 100$ (Table 1, Table 2, and Table 3).

Table 1. The average time to memory overflow ($p_1 = p_2 = p_{12} = q_1 = q_2 = q_{12} = pq_{12} = pq_{21} = 0.11$)

m	partition in half	one after another
4	13.759	13.219
6	21.208	22.872
8	31.888	33.595
10	45.508	45.143
100	3158.0	3297.0

Table 2. The average time to memory overflow, matrix multiplication ($p_1 = 0.071$, $p_2 = 0.108$, $p_{12} = 0.014$, $q_1 = 0.071$, $q_2 = 0.108$, $q_{12} = 0.014$, $pq_{12} = 0.013$, $pq_{21} = 0.013$)

m	partition in half	one after another
4	37.050	38.932
6	57.243	65.721
8	88.169	88.458
10	126.268	119.246
100	8878.300	9320.350

Table 3. Comparison of runtime to overflow, knapsack problem ($p_1 = 0.025$, $p_2 = 0.05$, $p_{12} = 0.002$, $q_1 = 0.025$, $q_2 = 0.05$, $q_{12} = 0.002$, $pq_{12} = 0.002$, $pq_{21} = 0.002$)

m	partition in half	one after another
4	101.310	112.718
6	156.527	187.609
8	242.151	241.730
10	346.735	329.609
100	24334.267	26129.358

Runtime of the system based on "One after another" method is compared with the runtime of the system, where shared memory is split in advance between deques so that each deque has $m/2$ units of memory. The used work-stealing strategy is stealing of one element.

Analyzing the results, one can make the following conclusion: for some memory sizes, the system built on the basis of the proposed method works

longer. In an example, for the matrix multiplication problem, the difference in the system runtime until overflow (where the memory of size $m = 6$ is used) is 8.5. This means the system runs by 8 operations longer if deques are located in the memory one after another in a circle. For the knapsack problem, it runs by 31 operations longer.

With the increase of memory size, the difference in runtimes of the systems also increases. In an example, for the memory size of $m = 100$ and for the matrix multiplication problem the system will run by 442 operations longer if deques are located one after another in a circle. For the knapsack problem, it runs by 1795 operations longer.

4. Conclusion

Mathematical and simulation models of the process of work with two work-stealing deques moving one after another in a circle were built. The numerical analysis of this method was carried out. Theoretical and practical input data were used in the experiments.

The mathematical model was built as a random walk inside an integer pyramid with reflecting and absorbing screens. The algorithm and the program for calculating the mean runtime of the system to overflow have been developed. The simulation model of the process was built. The described method has been patented [**14**].

For each task, 10 million simulation experiments were conducted, for a smaller number leads to discrepancies with the results of mathematical modeling. This number of experiments can be explained by the large variance of the random variable (the number of steps to overflow), but it is worth noting that in these problems it does not accurately determine the quality of the experiments. Since the meantime of work is maximized, any deviations from the mean value upward are desirable.

It can also be noted that the paper does not give the conventional formulation of the optimization problem. Here, the optimality criterion cannot be written analytically, and it is calculated algorithmically. It will not be possible to solve the system of differential equations in this problem, but in some other problems, such systems were solved numerically [**32**].

Proposed models, algorithms and programs for analyzing "One after another" method of work-stealing deques representation can be used in the development of operating systems, schedulers, memory managers and other system programs.

Using the built models one can choose (knowing probabilistic characteristics of deques in advance) the best method of representation of the

data structures, for example, from the two methods: classic sequential cyclic method or "One after another" method.

In [17] and [31] the mathematical models of optimal control of one and two stacks in two-level memory were proposed, in [32]—models of the optimal control of FIFO-queues. In practice, various architectures have hardware implemented a number of methods for managing stacks in two-level memory as an alternative to the classic cache memory [33]. In the paper, we have discussed the models of optimal control of two deques in single-level memory, but in the future, it will be important to build the models of optimal control of deques in two-level memory.

As our experience in software implementation has shown, the correct usage of cache memory in the parallel work-stealing load balancers is of great importance. For example, in the implementation proposed in [29], it was possible to reduce the overhead of the scheduler by 2.5 times and misses at the last cache level by 30% in comparison with the work-stealing schedulers by Intel TBB and Intel/MIT Cilk.

Therefore it is necessary to research and implement the optimal hardware implementations of deques, rather than trying to adapt to the universal implementations of cache memory.

References

[1] R.D. Blumofe, C.E. Leiserson. "Scheduling multithreaded computations by work stealing", *Journal of the ACM*, **5**:46 (1999), pp. 720–748. ↑20

[2] M. Herlihy, N. Shavit. – *The Art of Multiprocessor Programming*, Elsevier, 2008, 536 pp. ↑20

[3] R.D. Blumofe, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall, Y. Zhou. "Cilk: An Efficient Multithreaded Runtime System", *Journal of Parallel and Distributed Computing*, **37**:1 (1996), pp. 55-69. ↑20

[4] D. Leijen, W. Schulte, S. Burckhardt. "The Design of a Task Parallel Library", *ACM SIGPLAN*, **44**:10 (2009), pp. 227-242. ↑20

[5] O. Tardieu, H. Wang, H. Lin. "A work-stealing scheduler for X10's task parallelism with suspension", *ACM SIGPLAN*, **47**:8 (2012), pp. 267-276. ↑20

[6] G. Varisteas. *Effective cooperative scheduling of task-parallel applications on multiprogrammed parallel architectures. Doctoral Thesis in Information and Communication Technology*, Royal Institute of Technology, KTH, Stockholm, Sweden, 2015. ↑20

[7] D. Knuth. *The Art of Multiprocessor Programming*. V. 1, Addison-Wesley Professional, 1997. ↑20

[8] D. Hendler, N. Shavit. "Non-blocking Steal-half Work Queues", *ACM PODC*, 2002, pp. 280-289. ↑20

[9] A.V. Sokolov, A.V. Drac. "The Linked List Representation of n LIFO-Stacks and/or FIFO-Queues in the Single-Level Memory", *Information Processing Letters*, **13** (2013), pp. 832-835. ↑20,21

[10] E.A. Aksenova, A.A. Lazutina, A.V. Sokolov. "About the Optimal Methods of Representation of Dynamic Data Structures", *Review of applied and industrial mathematics*, **10**:2 (2003), pp. 375-376 (Russian). ↑20,21

[11] D. Hendler, Y. Lev, M. Moir, N. Shavit. "A Dynamic-Sized Nonblocking Work Stealing Deque", *Distributed Computing*, **18**:3 (2006), pp. 189-207. ↑20

[12] M. Mitzenmacher. "Analyses of Load Stealing Models Based on Differential Equations", *ACM SPAA*, 1998, pp. 212-221. ↑20

[13] A.V. Kalachev. *Multicore Architectures*, Moscow, BINOM, 2014 (Russian), 247 pp. ↑20

[14] E.A. Barkovsky, A.V. Sokolov. "A Way to Manage the Memory of a Computer System. No. 2647627", *Bulletin no. 8*, publ. 16.03.2018 (Russian). ↑21,28

[15] A.V. Sokolov. *Mathematical Models and Algorithms of the Optimal Control of Dynamic Data Structures*, Petrozavodsk, PetrSU, 2002 (Russian). ↑21

[16] E.A. Barkovsky, A.V. Sokolov. "Management Model for Two Parallel FIFO Queues Moving One after Another in Shared Memory", *Information and Control Systems*, 2016, no.1, pp. 65-73 (Russian). ↑21

[17] E.A. Aksenova, A.A. Lazutina, A.V. Sokolov. "Study of a Non-Markovian Stack Management Model in a Two-Level Memory", *Programming and Computer Software*, **30**:1 (2004), pp. 25-33. ↑21,29

[18] E.A. Aksenova, A.V. Sokolov. "The Optimal Implementation of Two FIFO-Queues in Single-Level Memory", *Applied Mathematics*, **2** (2011), pp. 1297-1302. ↑21

[19] E.A. Barkovsky, A.A. Lazutina, A.V. Sokolov. "The Model of Control of Two Work-Stealing Deques Moving One After Another in a Shared Memory", *Review of applied and industrial mathematics*, **25**:1 (2018), pp. 77 (Russian). ↑21

[20] D. Chase, Y. Lev. "Dynamic Circular Work-Stealing Deque", *ACM SPAA*, 2005, pp. 21-28. ↑21

[21] E.A. Barkovsky, A.V. Sokolov. "Probabilistic Model for the Problem of Optimal Control of Work-Stealing Deques with Various Strategies of Work-Stealing", *Strategies of Work-Stealing. Probabilistic methods in discrete mathematics*, 2016, pp. 11-13 (Russian). ↑21

[22] A.V. Sokolov, E.A. Barkovsky. "The Mathematical Model and The Problem of Optimal Partitioning of Shared Memory for Work-Stealing Deques", *Lecture Notes in Computer Science*, **9251** (2015), pp. 102-106. ↑21

[23] E.A. Aksenova, A.V. Sokolov. "Modeling of the Memory Management Process for Dynamic Work-Stealing Schedulers", *ISPRAS*, 2018, pp. 12-15. ↑21

[24] S. Mattheis, T. Schuele, A. Raabe, T. Henties, U. Gleim. "Work Stealing Strategies for Parallel Stream Processing in Soft Real-Time Systems", *Lecture Notes in Computer Science*, **7179** (2002), pp. 172-183. ↑21

[25] A.I. Adamovich, A.V. Klimov. "How to create deterministic by construction parallel programs? Problem statement and survey of related works", *Program systems: Theory and applications*, 2017, no.4, pp. 221-244 (Russian). ↑21

[26] E.A. Barkovsky, R.I. Kuchumov, A.V. Sokolov. "Optimal control of two deques in shared memory with various work-stealing strategies", *Program systems: Theory and applications*, 2017, no.1, pp. 83-103 (Russian). ↑21,22,26

[27] R.I. Kuchumov. "Implementation and Analysis of the Work-Stealing Task Scheduler", *Stochastic optimization in computer*, **12**:1 (2016), pp. 20-39 (Russian). ↑21,26

[28] E.A. Barkovsky. "Implementation of the Memory Manager in the Work-Stealing Scheduler", *Stochastic optimization in computer science*, **13**:1 (2017), pp. 3-12 (Russian). ↑22

[29] R. Kuchumov, A. Sokolov, V. Korkhov. "Staccato: Cache-Aware Work-Stealing Task Scheduler for Shared-Memory Systems", *Lecture Notes in Computer Science*, **10963**:1 (2018), pp. 91-102. ↑22,29

[30] J.G. Kemeny, J.L. Snell. *Finite Markov Chains*, Van Nostrand, 1969. ↑26

[31] E.A. Aksenova, A.V. Sokolov. "Optimal Control of Two Parallel Stacks in Two-Level Memory", *Discrete mathematics*, **19**:1 (2007), pp. 67-75 (Russian). ↑29

[32] A.V. Sokolov. "About the Optimal Caching of FIFO Queues", *Stochastic optimization in computer science*, **9**:2 (2013), pp. 72-88 (Russian). ↑28,29

[33] P.J. Koopman. *Stack Computers: The New Wave*, Ellis Horwood Ltd., 1989. ↑29

Received	28.10.2018
Revised	20.11.2018
Published	15.02.2019

Recommended by *p.h.d. Sergey A. Amelkin*

Sample citation of this publication:

Eugene Barkovsky, Anna Lazutina, Andrew Sokolov. "The Optimal Control of Two Work-Stealing Deques, Moving One After Another in a Shared Memory". *Program Systems: Theory and Applications*, 2019, **10**:1(40), pp. 19–32.

10.25209/2079-3316-2019-10-1-19-32

http://psta.psiras.ru/read/psta2019_1_19-32.pdf

The same article in Russian: 10.25209/2079-3316-2019-10-1-3-17

About the authors:

Eugene Aleksandrovich Barkovsky

Graduated from the Petrozavodsk State University in 2012. In 2015, graduated from the graduate school of the Institute of Applied Mathematical Research of the Karelian Research Center of the Russian Academy of Sciences (laboratory of information computer technologies). Author of 17 scientific publications on parallel dynamic data structures. Research interests: problems of optimal control of parallel dynamic data structures, work-stealing balancers.

0000-0001-9041-6453
e-mail: barkevgen@gmail.com

Anna Aleksandrovna Lazutina

Chief specialist of the Information Resources Department of the Informatization Department of the Moscow State University. Graduated from Petrozavodsk State University in 2004, Ph.D. (2006). Author of publications about optimal stack control problems. Research interests: applied mathematics and computer science, optimal control of dynamic data structures, controlled random walks, Markov chains.

0000-0001-7569-114X
e-mail: alazutina@yandex.ru

Andrew Vladimirovich Sokolov

Professor, a leading researcher in the Institute of Applied Mathematical Research of the Karelian Research Center of the Russian Academy of Sciences. Graduated from Leningrad University in 1974, Ph.D. (2006). The author of more than 100 scientific publications. Research interests: optimal control of dynamic data structures, optimal dynamic distribution of non-paged memory, controlled random walks, Markov chains, parallel computing, dynamic work-stealing balancers.

0000-0003-3787-7765
e-mail: sokavs@gmail.com

Эта же статья по-русски: 10.25209/2079-3316-2019-10-1-3-17