On the problem of optimal stack control

Dr. Andrew V. Sokolov

Department of Computer Science, University of Petrozavodsk

Lenin St., 33, Petrozavodsk, Republic of Karelia, 185640, Russia

E-mail: asokolov@mainpgu.karelia.ru

Abstract

This paper concerns issues related to building mathematical models and optimal algorithms of stack control in single- and two-level memory. These models were constructed as 1, 2 and 3-dimensional random walks. For solving this problem the Markov chain theory was applied. This research work was supported by the Russian Foundation for Fundamental Research, grant 95-01-00800.

1 Introduction

The stack concept [1] is used in the development of software and hardware for a wide range of problems. In the case of single-level memory, several methods of stack presentation in memory may be used for stack control. The connected presentation is the first method. In this case any number of stacks can coexist inside a shared memory area until the free memory list is exhausted. But on the other hand, this approach requires an additional link field for each stack element. The second method (Garvic's algorithm) uses the consecutive allocation of one stack after another. It is also possible to allocate stacks consequently and divide them into pairs of stacks growing towards each other [2]. This is the case we analyze here, as in [3], it has been shown that this method of controls is optimal.

2 Optimal control in single-level memory

In this paper it is assumed, that there are n stacks divided into pairs of stacks growing towards each other (the odd stack, if there is any, is assumed to grow towards the empty stack). Let q_i, p_i denote the probabilities of deletion and insertion information into the i-th stack (a process transfers from the state 0 into 0 with the probability q_i). Our task is to determine the optimal initial memory distribution and optimal memory redistribution in case one of the stacks overflows, where optimality is understood in the sense of maximizing the average functioning time until stack overflow. More exactly the problem will be stated for $n = 3$.

Suppose there are three stacks located in a memory area of volume m units. The pair of stacks growing towards each other occupies s memory units and $m - s$ memory units are left for the third stack. Let x_1, x_2 and x_3 denote the current stack heights. In this case the mathematical model is the three-dimensional random walk inside a prism with three reflecting barriers $x_1 = -1$, $x_2 = -1$, $x_3 = -1$ and two absorbing barriers $x_1 + x_2 = s + 1$, $x_3 = m - s + 1$.

The problem of finding optimal initial memory distribution consists in determining the value of s and numbering of the stacks (i.e. determining the number of the stack to be placed separately from the others), to maximize the average time of walk inside the prism until the absorption in its border, provided the process begins from the origin. In other words this task is reduced to choosing an optimal prism. These problems were solved using the Markov absorbing chain theory results.

The algorithms as well as the computational programs in C++ programming language for the case $n = 3$ and $n = 4$ were developed.

2.1 The case of three stacks, when only insertions are assumed ($q_i = 0$)

Here we'll consider the case of three stacks, when only insertions are allowed ($q_i = 0$). In this case we have the probability of moving out of the position (x_1, x_2, x_3) to the position $(x_1 + 1, x_2, x_3)$ — p_1, to the position $(x_1, x_2 + 1, x_3)$ — p_2, and to the position $(x_1, x_2, x_3 + 1)$ — p_3 where $p_1 + p_2 + p_3 = 1$.

Let $x = x_1 + x_2$, $y = x_3$, $p = p_1 + p_2$ and $q = p_3$. Then the

2-dimensional random walk in the integer lattice space, where $0 \le x \le s$, $0 \le y \le m - s$ is used as a mathematical model. The probability of moving out of the position (x, y) to the position $(x+1, y)$ is p, to the position $(x, y+1)$ is $q = 1 - p$.

The process starts from the state $(0, 0)$ and is absorbed at the lines $x = s + 1$ and $y = m - s + 1$.

The objective of our study is to find the value s, where the mean time of walk to the absorption would be maximal,

We have some methods for solution this problem.

1. One can apply the Markov chain theory.
2. One can solve the respective difference equation.
3. The combinatorial solution, when we calculate the number of paths from the point $(0, 0)$ to respective points.

The algorithms as well as the computational programs in C++ programming language for the all methods were developed. In Table 1 calculated probabilities of insertion into stack, which should be located separately q, optimal values of parameter s, and mean time until overflow with optimal values of parameters T with $m = 10$ are listed.

It is seen that if one probability of insertion into stack is very small comparing with some other one, then

$$s \approx p * m \ , \quad \text{or} \quad s \approx (p * m) - 1 \ ,$$

where $q = \min(p_1, p_2, p_3)$, $p = 1 - q$.

If the probabilities are approximately equal, then

$$s \approx p * m \ , \quad \text{or} \quad s \approx (p * m) - 1 \ ,$$

where $q = \max(p_1, p_2, p_3)$, $p = 1 - q$. In this case, as it was shown by calculations some variants are possible, q is equal to the mean value of probability. For example, if $m = 30$, $p_1 = 0.33$, $p_2 = 0.35$, $p_3 = 0.32$ then

if $q = 0.32$ then $s = 20$, $T = 27.60467$,

if $q = 0.35$ then $s = 19$, $T = 27.60499$,

if $q = 0.33$ we have optimal variant $s = 19$, $T = 27.61359$.

Table 1. *Results of computations*

p_1	p_2	p_3	q	s	T
0.8	0.2	0.0	0.0	10	11.0
0.8	0.19	0.01	0.01	10	10.466
0.8	0.18	0.02	0.02	9	10.12
0.6	0.3	0.1	0.1	8	9.66
0.5	0.3	0.2	0.2	7	9.397
0.5	0.24	0.26	0.24	7	9.448
0.5	0.25	0.25	0.25	7	9.442
0.4	0.3	0.3	0.3	7	9.313
0.333	0.333	0.333	0.333	6	9.297
0.35	0.35	0.3	0.35	6	9.321
0.35	0.33	0.32	0.35	6	9.321
0.36	0.32	0.32	0.36	6	9.328
0.38	0.31	0.31	0.38	6	9.329

If $p_1 = p_2 = p_3 = 1/3$, then s is not equal to $m * 2/3$, as it seems intuitively. For example, if $m = 1000$, optimal variant $s = 661$, but not at all $s = 667$.

These methods we can apply for four stacks.

3 Two-level memory

Now that we have examined how stack overflows and underflows occur in program execution, how should they be handled? Four possible ways of handling spills are [4]:

1. *A very large stack memory.* The simplest way to solve the stack problem is simply to assume that stack overflows will never happen. WISC CPU/16 uses this method with a stack size of 256 elements.

2. *Demand fed single-element stack manager.* To implement this strategy, the stack buffer is set up as a circular buffer with a head and tail pointer. A pointer to memory is also needed to keep track of the

top element of the memory-resident portion of the stack. Whenever a stack overflow is encountered, the bottom-most buffer-resident element is copied to memory, freeing a buffer location. Whenever an underflow is encountered, one element from memory is copied into the buffer. This technique has the appeal that the processor never moves a stack element to or from memory unless absolutely necessary, guaranteeing the minimum amount of stack traffic. RISC processors use this method for local scalars and procedures arguments management [5, 6, 7].

3. *Paging stack manager.* An alternative to the demand-fed strategy is to generate an interrupt on stack overflow and underflow, then use software to manage the stack spill. This approach uses less control hardware than the demand-fed method, but requires a stack buffer that is somewhat bigger to reduce the frequency of the interrupts. The general strategy used in this scheme is to have limit registers pointing to locations near the top and bottom of the stack buffer space. When an instruction causes the stack pointer to be less than underflow pointer, a half-buffer full of elements is copied from program memory. When an instruction exceeds the overflow pointer, a half-buffer full of elements is copied into program memory. The paging scheme allows arbitrarily sized sections of a large stack memory to be used by different procedures on a time-sliced basis. Because of this, the stack buffer appears as a section of special memory, not as a circular buffer. Therefore, in practice, a stack overflow actually involves copying a half-buffer of elements to memory, then relocating the other half-buffer to place it at the start of the stack buffer area. RTX2000 and RTX32P both use this method for stack management.

4. *An associative cache.* This method involves significant hardware complexity but does not provide any advantage over the previously mentioned methods for stack machines, since stack machines do not skip about much in accessing their stack elements.

The "demand" method always swaps one element and the "paging" method swaps half of the fast memory. We want to determine the optimal number x_0 of stack elements in the fast memory, if some probabilistic characteristics of stacks and time characteristics of memory are known.

3.1 Single stack in two-level memory

3.1.1 Case 1. Without taking into account the cost of swapping

We now consider the case of a single stack.

Without taking into account the cost of swapping, the problem reduces to the search of the initial state x_0 of the 1-dimensional random walk, that maximizes the expected walk time until absorption in the states -1 and $m+1$. The model reduces to the problem of gambler ruin, when the common capital of gamblers equals $m+2$ and the first gambler's capital equals x_0+1.

In the case of fixed information deletion probabilities (q) and insertion probabilities(p) there is an analytical solution of the respective difference equation for the game time [8] and the optimal value x_0 may be derived analytically and looks as follows:

$$x_0 = \begin{cases} \log_{\frac{q}{p}} \left[\dfrac{\left((q/p)^{m+2}-1\right) p}{(m+2)q\ln(q/p)} \right], & \text{if } p \neq q\ . \\ \dfrac{m}{2}, & \text{if } p = q = 1/2\ , \end{cases}$$

3.1.2 Case 2. Swapping costs are taken into account

If swapping costs are taken into account, then as an optimization criterion we can choose the average time of access to the top of stack. In the present paper we consider a generalization of the problem stated in [9] and present an optimization criterion in terms of cache memory theory [10].

Similar formulation of the problem was discussed in [11] as well. In this paper it has been proposed in overflowing to relocate k elements of fast top of stack into the secondary memory and to relocate the rest upper elements to the start of fast memory, but in emptying of top stack it has been suggested to relocate k elements from the secondary memory into the fast one.

Unfortunately, in modeling and in algorithm analyzing an inaccuracy was committed. The value D_k is the mean time until memory overflow or underflow is involved in the optimization criterion, if the process begins from the state k (the number of stack elements in fast top). But we fall into this state only after underflow, while after overflow we pass into the

state $m-k+1$ and, consequently, in the optimization criterion D_{m-k+1} must be used but not D_k. In our model the number of elements in fast memory, but not the number of relocated elements is used as a parameter, and the probability of overflow as well as of underflow are taken into consideration.

Let t_0 denote the access time to the top of the stack in fast memory, t_1—the transfer time of one element in fast memory, t_2—the access time to second-level memory, t_3—the time of swapping one element between different memory levels.

According to the architectural solutions of loss function in states [7], relocations can have different forms and, therefore, we'll consider some possible variants of stack top implementation.

1. For example, for "paging" method loss function, specified in states, takes the form

$$f(x) = t_0\ , \quad f(-1) = t_2 + t_3 * x_0\ ,$$
$$f(m+1) = t_2 + t_3 * (m+1-x_0) + t_1 * (x_0 - 1)\ .$$

   In this case we suppose that in underflowing of the top of stack x_0 elements are relocated from the secondary memory into the buffer, while in overflowing $m+1-x_0$ elements are kept in the secondary memory and x_0-1 elements are relocated into the start of the buffer. Then, after the insertion of a new element we have x_0 elements in the buffer, and the new phase begins and continues until the next overflow or underflow. We assume that, when elements are relocated, the loss is proportional to the number of relocated elements.

2. In overflowing, it may be considered that the relocation of elements in the buffer does not depend on the number of elements and takes a fixed time t. Then $f(m+1) = t_2 + t_3 * (m+1-x_0) + t$.

3. For the circular stack buffer $f(m+1) = t_2 + (m+1-x_0) * t_3$, since in contrast to the previous case, at stack overflow upper elements of stack are not relocated into the start of the buffer.

   Depending on the apparatus solutions, loss functions can have another form.

4. It can be assumed that the swapping between memory levels is implemented so that the time of relocation does not depend on the number of relocated elements. Then $f(-1) = f(m+1) = t_2$.

5. The implementation with copying is possible in background mode, and then $f(-1) = t_2 + t_3 * x_0$, $f(m+1) = 0$. In this case stack elements storage in memory is not required at overflow, but more complex equipment is necessary, and the access time to the top of stack increases significantly in fast memory.

There are some other possible variants of architectural solutions. For example, in [7] it was proposed for RISC processors to set up windows of variable size, which is determined during the run of procedure. The mathematical model can be described approximately as a random walk, since in this case the probability of relocation from the state x to $x - l$ (return from the procedure of length l) depends on the length of procedures in stacks, that is a non-Markov process takes place.

Now we consider the average coefficient of losses, which equals the ratio number of addresses to memory, called swapping, to a general number of addresses. For the stack memory this coefficient equals $1/T(x_0)$, where $T(x_0)$ is the expected walk time until absorption. The average time of access equals

$$F(x_0) = \frac{1}{T(x_0)} C(x_0) + \frac{T(x_0) - 1}{T(x_0)} t_0 = t_0 + \frac{C(x_0) - t_0}{T(x_0)} ,$$

where $C(x_0)$ is average losses for swapping in states -1 and $m+1$.

$$C(x_0) = p_{x_0,-1} f(-1) + p_{x_0,m+1} f(m+1) ,$$

where $p_{x_0,-1}$ is the probability that starting at x_0 we are absorbed in -1, $p_{x_0,m+1}$ is the probability that starting at x_0 we are absorbed in $m+1$.

The objective of our study is to find the value x_0, where the average time of access to the top of the stack $F(x_0)$ would be minimal.

1. Let $p = q = 1/2$. Then

$$p_{x_0,-1} = 1 - \frac{x_0 + 1}{m + 2} , \quad p_{x_0,m+1} = \frac{x_0 + 1}{m + 2} ,$$

$$T(x_0) = (x_0 + 1)(m + 1 - x_0) , \quad \text{see [8].}$$

While calculating the derivative of the function $F(x_0)$ on x_0, it can be shown, that the optimal value x_0 can be expressed using the following formulas for the parameters, which are of interest in practice. Since only integral numbers are physically meaningful, we have to choose the best value out of floor(x_0) and ceil(x_0) as an integral optimal value.

Variant 1.

$$x_0 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} ,$$

where $a = t_1 m + t_3$,
$b = -2t_0 m + 2t_1 m + 2t_2 m - 2t_3 m - 4t_0 + 4t_2 - 2t_3$,
$c = t_0 m^2 - t_2 m^2 + t_3 m^2 + 2t_0 m + t_1 m - 2t_2 m + 2t_3 m + t_3$.

Variant 2.

$$x_0 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} ,$$

where $a = t_3 + t$,
$b = -2t_0 m + 2t_2 m - 2t_3 m - 4t_0 + 4t_2 - 2t_3 + 2t$,
$c = t_0 m^2 - t_2 m^2 + t_3 m^2 + 2t_0 m - 2t_2 m + 2t_3 m + t_3 + t$.

Variant 3.

$$x_0 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} ,$$

where $a = t_3$,
$b = -2t_0 m + 2t_2 m - 2t_3 m - 4t_0 + 4t_2 - 2t_3$,
$c = t_0 m^2 - t_2 m^2 + t_3 m^2 + 2t_0 m - 2t_2 m + 2t_3 m + t_3$.

Variant 4.

$$x_0 = \frac{m}{2} .$$

Variant 5.

$$x_0 = m .$$

These are some results of calculation using given formulas, as an example. Let us take some constants from [5] and [11]. Let $t_0 = 1$, $t_2 = 30$, $t_3 = 16$, $t_1 = 1$, $t = 8$. Optimal values x_0 and corresponding values of mean access time $F(x_0)$ for the first three variants of top stack organization and several values m are presented in Tables 2–4.

Table 2. *Variant 1 of top stack organization*

m	8	16	32	64
x_0	3	5	10	16
$F(x_0)$	5.042	3.051	2.025	1.51
$F(m/2)$	5.12	3.08	2.04	1.518

Table 3. *Variant 2 of top stack organization*

m	8	16	32	64
x_0	3	6	12	23
$F(x_0)$	5.142	3.068	2.022	1.506
$F(m/2)$	5.2	3.086	2.028	1.508

Table 4. *Variant 3 of top stack organization*

m	8	16	32	64
x_0	3	6	13	25
$F(x_0)$	5.008	3.027	2.011	1.503
$F(m/2)$	5.040	3.037	2.014	1.504

It is seen from the tables that the mean access time to the top of stack tends to $t_0 = 1$, while the fast memory size increases. Notice that in all variants of architecture $x_0 \neq m/2$, although in practice, for example, in the "paging" method $x_0 = m/2$ is used.

2. If $p \neq q$ then a numerical solution of the problem is proposed.

In some applications, it is considered that a more adequate model of stack behavior is that, in which alongside the possibility of increase or decrease of the stack length, allowance is made for the stack length to be unchanged. In this situation the model changes are apparent. A particle transfers from the state x into $x-1$ with the probability p_1, into $x+1$ with the probability p_2, and into x with the probability p_3. It can be shown that the solutions of the discussed problems remain valid, and only p and q are replaced for p_2 and p_1.

3.2 Two stacks in two-level memory

This problem was originally posed by Knuth and formulated as follows. Let two stacks grow and collide inside a shared memory of volume m. In this case the location of their collision may be considered as a random variable. It is known that at each stage an element may be popped from the stack with the probability $1-p$ and some information may be pushed into one of the stacks with the probability p. Let $M(m,p)$ denote the expectation of the random variable $\max(k_1,k_2)$, where k_1 and k_2 are the heights of the stacks collided.

D. Knuth [1] posed the problem to construct the mathematical model of this process and to find the form of the function $M(m,p)$.

In [12, 13] the mathematical model of the process has been constructed as the two-dimensional random walk in a triangle with two reflecting boundaries and one absorbing boundary. The algorithm of computing $M(m,p)$ was proposed for the fixed parameter m. In order to solve the problem, the Markov chain theory was used.

In [14–18] the asymptotic behavior of stack sizes at the instant of the overflow, as well as the time till the memory overflow were investigated.

In the present paper we consider a generalization of Knuth's problem. We suppose, that in the case of memory overflow the process does not terminate. Instead, swapping with the second level memory occurs, and tops of stacks in fast memory transfer into some state, which becomes the starting position of subsequent computing. Our task is to find this state. The tendency is to have as few swappings as possible, but the swapping does occur when the memory is overflowed and the tops of the stacks are emptied. Hence, we have to search for the state, where the mean time till swapping would be maximum, provided the process starts from this state.

Let x_1, and x_2 denote the current stacks' heights. Then the 2-dimensional random walk in the integer lattice space, where $x_1 \geq 0$, $x_2 \geq 0$, $x_1 + x_2 \leq m$, is used as a mathematical model. The probability of moving out of the position (x_1, x_2) to the left is q_1, to the right—is p_1, downward—is q_2, upward—is p_2, and $p_1 + p_2 + q_1 + q_2 = 1$.

The objective of our study is to find the state $s_0 = (x_1^{(0)}, x_2^{(0)})$, where the mean time of walk to the absorption at the lines $x_1 + x_2 = m + 1$, $x_1 = -1$, $x_2 = -1$ would be maximal, provided the process starts from the state s_0.

This problem has been solved using the Markov absorbing chain theory results [19]. An algorithm of numbering the states was proposed, which made it possible to estimate the transition matrix corresponding to the Markov chain.

$$Q = \begin{pmatrix} \mathbb{0} & Q_m & \mathbb{0} & \cdots & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ P_m & \mathbb{0} & Q_{m-1} & \cdots & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & P_{m-1} & \mathbb{0} & \cdots & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \cdots & \cdots & \cdots & \ddots & \cdots & \cdots & \cdots \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \cdots & \mathbb{0} & Q_2 & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \cdots & P_2 & \mathbb{0} & Q_1 \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \cdots & \mathbb{0} & P_1 & \mathbb{0} \end{pmatrix}$$

where O is a zero matrix, Q_k is a matrix of size $k+1$ on k, P_k is a matrix of size k on $k+1$.

$$Q_k = \begin{pmatrix} q_1 & 0 & 0 & \cdots & 0 \\ q_2 & q_1 & 0 & \cdots & 0 \\ 0 & q_2 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & \cdots & q_1 \\ 0 & 0 & 0 & \cdots & q_2 \end{pmatrix}$$

$$P_k = \begin{pmatrix} p_1 & p_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & p_1 & p_2 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & p_1 & p_2 \end{pmatrix}$$

The algorithm as well as computational programs were developed for solving this problem. Since the program requires finding the inverse of the matrix of dimension m^2, special methods of storing sparse matrixes were involved. Some results of numerical experiments were obtained. In a symmetrical case for fixed probabilities $p_1 = p_2 = q_1 = q_2 = 0.25$, it was found that $x_1^{(0)} \approx x_2^{(0)} \approx \frac{m}{3}$.

4 Conclusion

In practice, the probability characteristics of stacks depend on numerous factors. In the majority of applications, at the beginning of work the insertion into a stack is more probable, in the middle of work the insertion

and the deletion are equally probable, and at the end deletion is more probable, as the work starts and finishes with an empty stack. For the implementation of optimal algorithms of stack control, the investigation of stack behavior is required with different input data. Such investigation has been carried out in creating RISC processors. Six specific programs (fractal generator, Fort compiler, Hanoi towers, N-queens problem) were analyzed for "demand" method [4]. In these programs maximal stack size was equal to 52 elements for Hanoi towers. In [20] stack behavior for an implementation of procedures was investigated in the interests of the proceject Bell Labs C Machine. In their experiments the stack depth reached 1106 words using compiler PDP-11. Certainly, it would be interesting to carry on the experiment and to compare practical and theoretical optimal methods of stack swap implementation using concrete programs.

Although it is difficult to say, what nearest prospects computers with stack architecture possess, their evident profits in the range of real time embedded control applications such as small size, high processing speed and excellent support of interrupt handling allow to hope that the concept of stack architecture in its spiral development would be embodied in new processors. In this connection, it should be noted that the virtual Java-machine, for example, is a stack machine [21]. In any case, stacks due to the fundamentality of recursion principle in informatics, will always be of importance for the implementation of the software and hardware support of computers.

Acknowledgments

The author would like to thank S. S. Lavrov, E. A. Zhogolev, P. J. Koopman and V. V. Antonov for stimulating discussions.

References

[1] D. E. Knuth, *The art of computer programming.* Vol. 1. Addison-Wesley, Reading, MA, 1976.

[2] S. Avierri, *Paired sequential lists in a memory interval.* Inform. Process Lett, No 8, 9, 10, 1979.

[3] A. Sokolov, *Optimal Dynamic distribution of non-page virtual memory.* PhD Thesis, LSU, 1981. (in Russian)

[4] P. J. Koopman, *Stack Computers*. Ellis Horwood, 1989. `http://www.cs.cmu.edu/~koopman/stack_computers/`

[5] Y. Tamir, C. Sequin, *Strategies of control register's file*. IEEE Trans. Comput. c-32(11), 1983. p. 977.

[6] D. A. Patterson, C. H. Sequin, *Reduced Instruction Set VLSI Computer*. Proceedings of the 8th Symposium on Computer Architecture. No 6, pp. 72–86, 1980.

[7] M. G. H. Katevanis, C. H. Sequin, D. A. Patterson, and R. W. Sherburne, *RISC:Effective Architectures for VLSI Computer*. VLSI Electronics. Microstructure Science, vol. 14, (ed. N. G. Einspruch), 1986.

[8] W. Feller, *An introduction to probability theory and its application*. Vol. 1, Wiley, New York, 1964.

[9] V. Mazalov, A. Sokolov, *About optimal dynamic storage allocation*. Control in dynamic systems, Leningrad, 1979. (in Russian)

[10] T. Kohonen, *Contents-Addressable Memories*. Springer-Verlag, 1980.

[11] Hasegava M., Shigei Y., *High-speed top-of-stack scheme for VLSI processor: a management algorithm and its analysis*. Proceedings of the 12th Symposium on Computer Architecture, pp. 48–54, 1985.

[12] A. Sokolov, *About storage allocation for implementing two stacks*. Automation of experiment and data processing, Petrozavodsk, pp. 65–71, 1980. (in Russian)

[13] A. Sokolov, *On the problem of optimal stack control in two level memory*. Probabilistic methods in discrete mathematics, Proceedings of the Fourth International Petrozavodsk Conference VSP, Utrecht, The Netherlands, pp. 349-351, 1997.

[14] A. C. Yao, *An analysis of a memory allocation scheme for implementating stacks*. SIAM J. Computing. No 10, pp. 398–403, 1981.

[15] P. Flajolet, *The evolution of two stacks in bounded space and random walks in a triangle*. Lec. Notes Computer Sci., vol. 223, pp. 325–340, 1986.

[16] G. Louchard and R. Schott, *Probabilistic analysis of some distributed algorithms.* Lect. Notes Computer Sci., vol. 431, pp. 239–253, 1990.

[17] G. Louchard, R. Schott, M. Tolley, P. Zimmermann, *Random walks, heat equation and distributed algorithms.* J. Comput. Appl. Math., vol. 53, pp. 243–274, 1994.

[18] R. S. Maier, *Colliding Stacks: A Large Deviations Analysis.* Random Structures and Algorithms, No 2, pp. 379–421, 1991.

[19] Kemeny J. G., Snell J. L., *Finite Markov Chains.* Van Nostrand, Princeton, New Jersey, 1960.

[20] D. R. Ditzel., H. R. McLellan, *Register Allocation for Free: The C Machine Stack Cache.* Proc. Symp. Archit. Support Progr. Lang. Oper. Systems, ACM 0-89791-066-4 82/03/0057, pp. 48–56, 1982.

[21] A. Taivalsaari, *Implementing a Java Virtual Machine in the Java Programming Language.* Sun microsystems, Technical Report Series, March 1998.