ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
INFORMATION TECHNOLOGY AND TELECOMMUNICATIONS

УДК 004.021, 004.42, 004.074

DOI: 10.21822/2073-6185-2025-52-2-90-97 Оригинальная статья/ Original article

Новый метод определения характеристик блочных алгоритмов

В.А. Егунов
Волгоградский государственный технический университет,
400005, г.Волгоград, пр.им. Ленина, 28, Россия

Резюме. Цель. В области высокопроизводительных вычислений большое значение имеет оптимизация программных систем под конкретные вычислительные архитектуры. Целью исследования является совершенствование методов определения параметров блочных алгоритмов. **Метод.** Исследование основано на методологии компьютерного моделирования и автоматизированного проектирования в технике и технологиях, включая постановку, формализацию и типизацию проектных процедур и алгоритмов. **Результат.** Предложен аналитический подход к повышению эффективности проектируемых программных систем, в рамках которого рассмотрен метод определения параметров блочных алгоритмов. Преимуществом данного метода является возможность его реализации на основе статического анализа исходного кода программы на начальных этапах проектирования, полученные параметры могут быть использованы в процессе оптимизации. Проведены вычислительные эксперименты, подтверждающие эффективность предложенного метода. **Вывод.** Предложенный метод позволяет определять размеры блоков, при которых на определенном уровне кеш-памяти не будет генерироваться кеш-промахов или количество этих промахов будет оставаться на приемлемом уровне. Данный метод может использоваться на начальных этапах проектирования программных систем, не прибегая к динамическому анализу приложения, включающему в себя этапы получения исполняемого кода приложения и анализ его с помощью специальных приложений, в данном случае используется статический анализ исходного кода программы.

Ключевые слова: кеш-память, оптимизация, эффективность программного обеспечения, кеш-промах, кеш-попадание, ассоциативность кеш-памяти, блочный алгоритм, проектирование программ

Для цитирования: В.А. Егунов. Новый метод определения характеристик блочных алгоритмов. Вестник Дагестанского государственного технического университета. Технические науки. 2025;52(2):90-97. DOI:10.21822/2073-6185-2025-52-2-90-97

A new method for determining the characteristics of block algorithms

V.A. Egunov
Volgograd State Technical University,
28 Lenin Ave., Volgograd 400005, Russia

Abstract. Objective. In the field of high-performance computing, optimization of software systems for specific computing architectures is of great importance. The aim of the study is to improve the methods for determining the parameters of block algorithms. **Method.** The study is based on the methodology of computer modeling and automated design in engineering and technology, including the formulation, formalization and typification of design procedures and algorithms. **Result.** An analytical approach to improving the efficiency of designed software systems is proposed, within the framework of which a method for determining the parameters of block algorithms is considered. The advantage of this method is the possibility of its implementation based on static analysis of the source code of the program at the initial stages of design; the obtained parameters can be used in the optimization process. Computational experiments confirming the efficiency of the proposed method were conducted. **Conclusion.** The proposed method allows

determining the block sizes at which no cache misses will be generated at a certain level of cache memory or the number of these misses will remain at an acceptable level. The method can be used at the initial stages of designing software systems without resorting to dynamic analysis of the application; in this case, static analysis of the program source code is used.

Keywords: cache memory, optimization, software efficiency, cache miss, cache hit, cache associativity, block algorithm, program design

For citation: V.A. Egunov. A new method for determining the characteristics of block algorithms. Herald of the Daghestan State Technical University. Technical Sciences. 2025; 52(2):90-97. (In Russ) DOI:10.21822/2073-6185-2025-52-2-90-97

Введение. Разработке эффективных программных систем (ПС) уделяется большое внимание исследователей. В течение последних десятилетий активно ведутся исследования в области создания новых методов проектирования и оптимизации ПС. Особое внимание уделяется этому вопросу в связи с ростом высокопроизводительных вычислений (ВПВ), которые связывают с обработкой больших массивов информации и ростом объемов вычислений. Примером ПС, непрерывно подвергающимся оптимизации на протяжении последних десятилетий, являются различные реализации интерфейса Basic Linear Algebra Subroutines (BLAS), который стал стандартом интерфейса библиотек, реализующих операции линейной алгебры [1,2]. Существует множество реализаций BLAS. Некоторые из них распространяются как библиотеки с открытым исходным кодом, например, OpenBLAS [3]. Другие распространяются в виде проприетарных библиотек, например IntelMKL [4]. Операции линейной алгебры можно считать классическим примером того, как можно значительно повысить эффективность ПС, снизив время вычислений, одновременно снижая требования к производительности вычислительной системы (ВС), за счет применения блочной оптимизации. Суть данного метода оптимизации заключается в разбиении большой матрицы на блоки (подматрицы) таким образом, чтобы все необходимые для выполнения очередного этапа преобразования блоки помещались в кеш-память. Данный метод показывает хорошую эффективность, однако, для его применения необходимо решить ряд проблем, связанных с преобразованием исходного алгоритма.

Постановка задачи. В области ВПВ большое значение имеет оптимизация ПС под конкретные вычислительные архитектуры. Существуют различные подходы решения данной задачи. Одним из подходов является разработка версий ПС, оптимизированных для конкретных вычислительных архитектур, как это сделано, например, в Intel MKL. Другим подходом к решению данной задачи является настройка ПС для получения наилучшей реализации на конкретной вычислительной архитектуре. Процедура настройки заключается в поиске наилучшей комбинации оптимизационных настроек или параметров. Цель подбора оптимального набора параметров – получение максимально эффективной ПС [5, 6].

Параметризованные алгоритмы при этом должны проектироваться таким образом, чтобы изменение параметров не влияло на логику работы ПС, т.е. ПС должна выдавать одни и те же результаты при задании различных параметров на одинаковых наборах входных данных. Различные наборы параметров должны влиять только на эффективность получаемой ПС в целевой ВС.

Предполагается ручной и автоматический варианты настройки алгоритмов. Ручная настройка является традиционным, известным и широко используемым методом оптимизации. Суть метода заключается в том, что специалист в области ВПВ подбирает параметры предложенных алгоритмов, создавая для каждой целевой архитектуры новую реализацию программы. Качество полученной программы и время, потраченное на ее разработку, зависят от квалификации специалиста. Автоматическая настройка представляет собой процесс, когда библиотека оптимизирует сама себя [7]. Примером может служить библиотека Automatically Tuned Linear Algebra Software (ATLAS), в которой для создания вычислитель-

ных ядер используется процедура итеративной компиляции с поиском оптимальных параметров настройки алгоритмов. Известны работы по разработке математических моделей и методов, которые применяются в процессе оптимизации ПС. В качестве примера модели, используемой для настройки параметров алгоритмов, можно привести модель целевой архитектурной платформы, описанной в работе [8], позволяющая вычислять значения параметров вместо перебора их значений и тестирования полученных программ на целевых архитектурах. Также необходимо отметить работы [9-11], на которые в определенной степени опирается настоящее исследование, в которых описаны методы повышения эффективности подсистемы памяти и векторизации вычислений в процессе получения эффективных реализаций BLAS операций.

Важным приемом оптимизации ПС является разработка блочных алгоритмов реализации известных преобразований, примером здесь может служить матричное умножение. Размер блока необходимо выбирать таким образом, чтобы минимизировать количество кеш-промахов в процессе выполнения вычислений. И важным здесь является следующий вопрос: «Какой максимальный размер блока, при котором генерируются только «холодные» промахи и не происходит вытеснения строк из кеш-памяти ?». Ответ на данный вопрос является крайне важным для задания значений параметров блочных алгоритмов, которые могут меняться в зависимости от характеристики целевой ВС.

Методы исследования. В современных микропроцессорах (МП) кеш-память имеет множественно-ассоциативную организацию, при которой кеш-память делится на несколько банков, каждый из которых работает как кеш с прямым отображением. Строка основной памяти может быть отображена не в одну конкретную строку кеша, а в некоторое множество строк, т.е. в один из банков. При последовательном обращении по адресам основной памяти, такая организация кеш-памяти приводит к равномерному заполнению множеств кеша. Однако, при использовании других шаблонов доступа, ситуация меняется. Рассмотрим общий случай регулярного доступа к данным, которые не расположены в памяти непрерывно (рис.1).

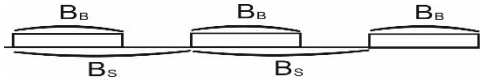

Рис. 1 – Схема расположения данных
Fig. 1 – Data layout diagram

Данные расположены регулярно, но не непрерывно:длина каждого шаблона составляет B_S байт; из каждого шаблона считывается B_B байт. В данном случае заполнение кеш-памяти не будет равномерным и будет зависеть от соотношений характеристик шаблона и кеш-памяти.

Пусть характеристики шаблона, выраженные в единицах кеш-линий, имеют значения B_S=7, B_B=5, количество множеств кеш-памяти N=4. Схема расположения данных будет выглядеть следующим образом (рис. 2).

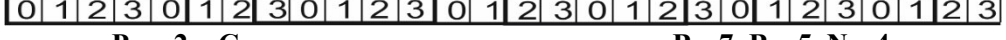

Рис. 2 – Схема расположения данных при B_S=7, B_B=5, N_S=4.
Fig. 2 – Data arrangement diagram for BS=7, BB=5, NS=4.

Числами от 0 до 3 здесь обозначены номера множеств кеш-памяти, в которые будут отображаться данные строки основной памяти. В каждом шаблоне считываются пять строк (обведены) и две не считываются (не обведены). Видно, что множества заполняются неравномерно и промахи начнут генерироваться до полного заполнения кеш-памяти из-за попытки помещения данных в заполненные множества.

Сформулируем теорему о заполняемости множеств, следствия из которой станут основой для метода определения размера блока. Пусть L и M – положительные взаимно простые числа, есть последовательность чисел (список) A, формируемая как конкатенация последовательностей (списков) A’ длины L со следующей структурой (1)

$$A' = [0, \dots, L-1] \quad (1)$$

Пусть также есть список B’ длины L, содержащий одну единицу в позиции k, остальные нули. Пусть есть упорядоченное множество B мощности M, первоначально содержащее нулевые значения. Пусть осуществляется обход списка A, для каждого элемента вычисляется функция f (2), модифицирующая элементы B.

$$f: B(i\%M) = B(i\%M) + B'(i\%L) \quad (2)$$

где i – номер элемента A.

Тогда существует конечное число S пройденных элементов A, после которых все элементы B будут содержать единицы, причем S = НОК(L, M), т.е. S=L*M в данном случае. Проиллюстрируем теорему на примере. Пусть L=7, M=3. Тогда при длине списка A, равной 7*3=21, k=0 получим следующие числовые ряды (рис.3).

A’	[	0	1	2	3	4	5	6	]													
i	[	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	]
A	[	**0**	1	*2*	**3**	4	*5*	**6**	0	*1*	2	**3**	*4*	**5**	6	*0*	**1**	2	*3*	**4**	5	*6*	]
B’	[	1	0	0	0	0	0	0	]														
B	[	1	1	1	]																		

Рис. 3 – Числовые ряды, иллюстрирующие теорему
Fig. 3 – Numerical series illustrating the theorem

В списке A элементы, использующиеся для модификации нулевого элемента B, выделены жирным на сером фоне, использующиеся для модификации первого элемента B выделены подчеркиванием, использующиеся для модификации второго элементы B, выделены курсивом. Можно сформулировать несколько следствий из данной теоремы.

Если в условии к теореме длина списка равна S=L*M, список B’ содержит единицы в v позициях, в остальных содержит нули, v<=L, тогда после прохода A все элементы B будут содержать значение v. Если L и M не являются взаимно простыми числами, тогда сформировать равномерно заполненное одинаковыми значениями множество B в соответствии с условием теоремы невозможно. Максимальная длина списка при этом также равна S = НОК(L, M).

Интерпретация теоремы для подсистемы памяти ВС:

L – длина шаблонов (шаг в памяти в элементах кеш-линий).;

M – число множеств в множественно-ассоциативном кеше;

B’ – указывает на то, какие элементы в шаблонах кешируются;

B – множества кеша; значения элементов соответствуют количеству заполненных банков в конкретном множестве;

A - память.

Таким образом, для определения максимального размера блока, не вызывающего кеш-промахов, необходимо определить: характеристики анализируемого уровня кеш-памяти (в частности, определить количество множеств в каждом банке M); характеристики шаблона доступа к памяти (длину шаблона в кеш-линиях L, количество кеш-линий в шаблоне, к которым производится обращение – количество единиц в множестве B’); S=НОК(L, M); с учетом степени ассоциативности определить максимальный размер блока.

Обсуждение результатов. Был проведен вычислительный эксперимент, который подтверждает сделанные ранее выводы. Эксперимент проводился на вычислительной системе, в состав которой входит МП 11th Gen Intel(R) Core(TM) i9-11900 @ 2.50GHz, который имеет следующие характеристики подсистемы памяти: - кеш L1D имеет объем 48 кБ и степень ассоциативности 12; кеш L2 имеет объем 512 кБ и степень ассоциативности 8; длина кеш-линии составляет 64 байта. Для проведения вычислительного эксперимента использовался язык программирования C, тип данных double (размер переменной составляет 8 байт).

Выберем разные длины шаблона, проведем аналитический расчет максимальной длины блока, далее проведем вычислительный эксперимент для проверки результатов.

Количество множеств в банках кеш-памяти МП составляет 64 в L1D и 1024 в L2, будем менять значение длины блока L, в опыте используем одну кеш-строку из всего шаблона (рис. 4).

Рис. 4 - Структура блока данных
Fig. 4 - Data block structure

Для этого при выделении памяти под массив учитываем выравнивание, также учитываем, что в одну строку кеша помещается 8 элементов double. В табл. 1 приведены характеристики двух шаблонов доступа в память, рассчитанные для кеша L2 в соответствии с предложенным методом.

Таблица 1 – Характеристики шаблонов доступа в память
Table 1 – Characteristics of memory access patterns

L, cl (d / b)	S=HOK(L,M)	c=S / L	N	S_B (d / b)	%
1024 (8192 / 64 кБ)	1024	1	8	65536 / 512Кб	0,098
512 (4096 / 32кБ)	1024	2	16	65536 / 512Кб	1,96

В первом столбце приведен размер шаблонов в кеш-линиях (cl), а также в элементах double (d) и байтах (b). Во втором столбце приводится значение наименьшего общего кратного для значений L и M (L2), которое в дальнейшем используется для расчета максимальной длины блоков. Длины шаблонов выбирались таким образом, чтобы заполнялось только одно множество в кеш-памяти каждого уровня. В третьем столбце приводится количество шаблонов, которые будут использоваться для размещения данных в одном банке L2. В четвертом столбце приводится количество шаблонов, непрерывно расположенных в памяти, которые помещаются в кеш-память L2, не вызывая промахов. В пятом столбце приводится общий размер блока в элементах double и байтах, В последнем столбце приводится процент заполнения кеш-памяти при использовании данного шаблона доступа до начала вытеснения строк. В результате были получены результаты, представленные на рис. 5 и 6. На рис. 5 представлен результат моделирования для L=8192d, на рис. 6 – L=4096d.

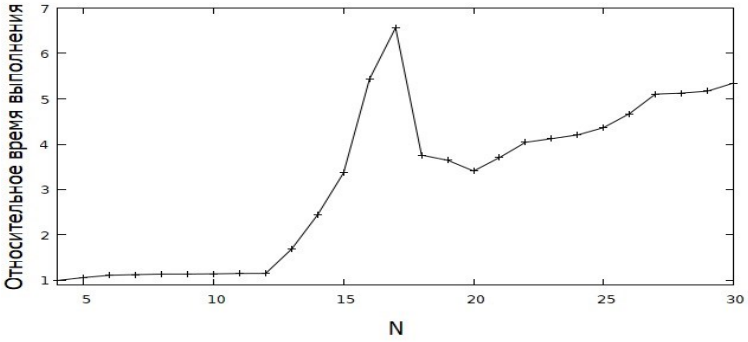

Рис. 5 – Результат моделирования для L=8192d
Fig. 5 – Simulation result for L=8192d

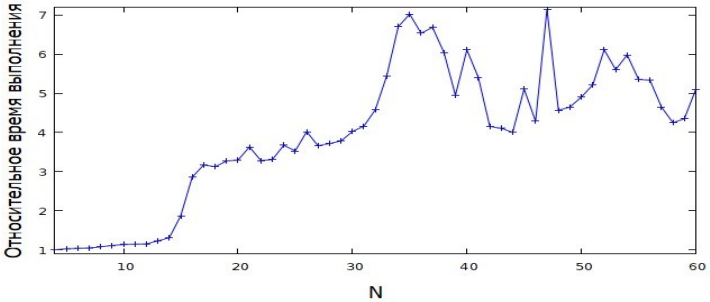

Рис. 6 - Результат моделирования для L=4096d
Fig. 6 - Simulation result for L=4096d

Измерения осуществлялись следующим образом. На первом этапе осуществлялся многократный доступ к элементам массива, размер которого совпадал с размером блока, состоящим из N шаблонов. Далее, для каждого измерения вычислялось среднее время обработки одного шаблона. На последнем этапе для каждого размера блока вычислялось относительное время выполнения, которое определялось как отношение среднего времени обработки шаблона к среднему времени обработки шаблона при N=4, с этого значения осуществлялось построение графика. При значении N, меньшем четырех, результаты оказываются зашумленными в связи со снижением вычислительной нагрузки и увеличением влияния длительности вспомогательных операций, например, операторов цикла.

Относительно полученных результатов можно сделать вывод, что они подтверждают сделанные аналитические оценки, при этом стоит отметить следующее. Промахи в L2 оказывают гораздо большее влияние на время выполнения программы, чем промахи в L1. Для кеша L2 промахи начинаются, начиная с N, приведенного в табл. 1, достигая максимума при размере блока, равным двойному размеру кеш-памяти (N = 16 на рис. 5 и N = 32 на рис. 6). Особенно хорошо данный эффект заметен на рис. 6. Т.к. используемые длины шаблонов подбирались таким образом, чтобы демонстрировать промахи в кешах обоих уровней, на этих рисунках можно наблюдать и промахи L1 при N=12 (степень ассоциативности кеша L1D). На рис. 5 эффект более заметен, т.к. на рис. 6 замедление при N=12 переходит в более значимое замедление при N=16, вызванное промахами в L2.

Третий эксперимент связан с анализом шаблонов, размер которых является взаимно простым числом с количеством множеств кеш-памяти (1024 в L2). Было выбрано значение 17 кеш-строк (136 в элементах double). Результат эксперимента приведен на рис. 7.

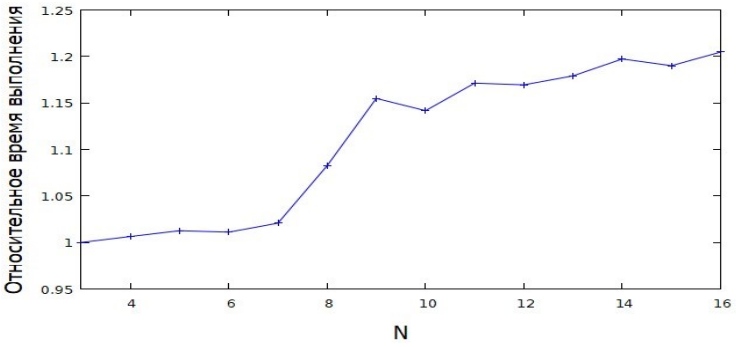

Рис. 7 - Результат эксперимента для размера шаблона, взаимно простого с размером множества L2

Fig. 7 - Experimental result for a pattern size that is relatively prime to the size of the L2 set

Эксперимент проводился следующим образом. Фиксировался размер шаблона (17 кеш-линий или 136 double). Далее варьировалось заполнение шаблона, параметр N на графике соответствует текущему заполнению (от 1 до 16 кеш-линий). Относительное время выполнения считалось аналогично графикам на рис. 5 и 6. В соответствии со следствиями из приведенной теоремы, при подобном размере шаблона кеш-промахи не будут генерироваться при заполнении, не превышающем степень ассоциативности кеш-памяти. На графике (рис. 7) видно, что рост времени выполнения наблюдается при N=8, что соответствует степени ассоциативности L2. Результат эксперимента также полностью подтверждает сделанные ранее выводы.

Вывод. Представлен метод определения максимального размера блока данных в блочных алгоритмах, при котором гарантируется эффективная работа подсистемы памяти ВС. Другими словами, предложен метод, позволяющий определять размеры блоков, при которых на определенном уровне кеш-памяти не будет генерироваться кеш-промахов или количество этих промахов будет оставаться на приемлемом уровне.

Данный подход удобен тем, что может использоваться на начальных этапах проектирования ПС, не прибегая к динамическому анализу приложения, включающему в себя этапы получения исполняемого кода приложения и анализ его с помощью специальных

приложений, в данном случае используется статический анализ исходного кода программы. В работе проанализирован доступ к одному массиву, однако, подобный подход может быть использован для оценки максимальных размеров нескольких массивов, одновременно используемых в приложении.

Зависимости, представленные на графиках, в большей степени отражают процессы, происходящие в кеш-памяти L2, чем в L1D. Это связано с большей разницей в скорости работы между L3 и L2, чем между L2 и L1D, таким образом, промахи в L2 приводят к большим временным задержкам, чем промахи в L1.

Кроме того, в процессе анализа использовался регулярный доступ к памяти: шаблоны обрабатывались последовательно друг за другом, внутри шаблона данные также обрабатывались последовательно. Это приводило к тому, что механизм предвыборки данных, существующий во всех современных МП, сглаживал проблемы, связанные с кеш-промахами (например, на рис. 5 и 6 не наблюдается резкое увеличение времени обработки при генерации кеш-промахов в L2 при N=8, хотя увеличение времени обработки имеет место). При другом порядке обращения к элементам блоков, например, при случайном порядке обработки, эффекты были бы более акцентированными. Однако, был выбран регулярный порядок обработки, т.к. в большинстве реальных задач используется именно он. Проведенные вычислительные эксперименты подтвердили выводы, полученные в соответствии с предложенным методом и доказали его эффективность.

Благодарности. Исследование выполнено за счет гранта Российского научного фонда № 25-21-20073 (https://rscf.ru/project/25-21-20073/) и гранта администрации Волгоградской области.

Acknowledgments. The study was supported by the Russian Science Foundation grant No. 25-21-20073 (https://rscf.ru/project/25-21-20073/) and a grant from the Volgograd Region Administration.

Библиографический список:

1. Lawson C.L., Hanson R.J., Kincaid D.R., Krogh F.T. Basic Linear Algebra Subprograms for Fortran Usage //ACM Transactions on Mathematical Software. 1979. Vol. 5, no. 3. P. 308-323. DOI: 10.1145/355841.355847.
2. Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J. An Extended Set of FORTRAN Basic Linear Algebra Subprograms//ACM Transactions on Mathematical Software. 1988. Vol. 14, no. 1. P. 1-17. DOI: 10.1145/42288.42291.
3. Xianyi Z., Qian W., Yunquan Z. Model-driven level 3 BLAS performance optimization on Loongson 3A processor//2012 IEEE 18th International Conference on Parallel and Distributed Systems (Singapore, December 17—19, 2012). Boston, Massachusetts, USA, IEEE Xplore Digital Library, 2012. P. 684—691. DOI: 10.1109/ICPADS.2012.97
4. Intel Math Kernel Library Reference Manual. URL: https://www.bu.edu/tech/files/2010/02/mklman61.pdf (дата обращения: 06.02.2025).
5. Goto K., Van De Geijn R. High-Performance Implementation of the Level-3 BLAS // ACM Transactions on Mathematical Software. 2008. Vol. 35, no. 1. P. 4:1—4:14. DOI: 10.1145/1377603.1377607.
6. Goto K., Geijn R.A. Anatomy of High-Performance Matrix Multiplication // ACM Transactions on Mathematical Software. 2008. Vol. 34, 158 no. 3. P. 12:1—12:25. DOI: 10.1145/1356052.1356053.
7. K. Czarnecki, U.W. Eisenecker, R. Glück, D. Vandevoorde, and T.L. Veldhuizen. Generative programming and active libraries. In Selected Papers from the International Seminar on Generic Programming, pages 25–39, London, UK, 2000. Springer-Verlag.
8. Low T.M., Igual F.D., Smith T.M., Quintana-Orti E.S. Analytical Modeling Is Enough for High-Performance BLIS // ACM Transactions on Mathematical Software. 2016. Vol. 43, no. 2. P. 12:1-12:18. DOI: 10.1145/2925987.
9. Kravets A.G., Egunov V. The software cache optimization-based method for decreasing energy consumption of computational clusters //Energies. – 2022. – Т. 15. – №. 20. – С. 7509..
10. Егунов, В.А. Метод улучшения стратегии кеширования для вычислительных систем с общей памятью / В.А. Егунов, А.Г. Кравец // Программная инженерия. - 2023. - Т. 14, № 7. - С. 329-338.
11. Егунов, В.А. Новый метод повышения эффективности векторизации операций BLAS / В.А. Егунов, А.Г. Кравец // Информационные технологии. - 2024. - Т. 30, № 6. - С. 318-328.

References:

1. Lawson C.L., Hanson R.J., Kincaid D.R., Krogh F.T. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*. 1979;5(3):308—323. DOI: 10.1145/355841.355847.
2. Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J. An Extended Set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*. 1988;14(1):1-17. DOI: 10.1145/42288.42291.
3. Xianyi Z., Qian W., Yunquan Z. Model-driven level 3 BLAS performance optimization on Loongson 3A processor // 2012 IEEE 18th International Conference on Parallel and Distributed Systems (Singapore, December 17—19, 2012). Boston, Massachusetts, USA, IEEE Xplore Digital Library, 2012:684-691. DOI: 10.1109/ICPADS.2012.97
4. Intel Math Kernel Library Reference Manual. URL: https://www.bu.edu/tech/files/2010/02/mklman61.pdf (date of request: 06.02.2025).
5. Goto K., Van De Geijn R. High-Performance Implementation of the Level-3 BLAS. *ACM Transactions on Mathematical Software.* 2008;35(1):4:1-4:14. DOI: 10.1145/1377603.1377607.
6. Goto K., Geijn R.A. Anatomy of High-Performance Matrix Multiplication. *ACM Transactions on Mathematical Software*. 2008;34(158(3)):12:1—12:25. DOI: 10.1145/1356052.1356053.
7. K. Czarnecki, U. W. Eisenecker, R. Glück, D. Vandevoorde, and T. L. Veldhuizen. Generative programming and active libraries. In Selected Papers from the International Seminar on Generic Programming, pages 25–39, London, UK, 2000. Springer-Verlag.
8. Low T.M., Igual F.D., Smith T.M., Quintana-Orti E.S. Analytical Modeling Is Enough for High-Performance BLIS. *ACM Transactions on Mathematical Software*. 2016; 43(2):12:1—12:18. DOI: 10.1145/2925987.
9. Kravets A.G., Egunov V. The software cache optimization-based method for decreasing energy consumption of computational clusters. *Energies*. 2022;15(20):7509.
10. Egunov V.A. A method for improving caching strategies for shared memory computing systems / Egunov V.A., Kravets A.G.. *Software Engineering*. 2023;14(7): 329-338. (In Russ.)
11. Egunov V.A. A new method to increase the efficiency of vectorization of BLAS operations / Egunov V.A., Kravets A.G. *Information technology*. 2024;30(6): 318-328. (In Russ.)

Сведение об авторе:

Виталий Алексеевич Егунов, кандидат технических наук, доцент; доцент кафедра «Электронно-вычислительные машины и системы»; vegunov@mail.ru

Information about author:

Vitaly A. Egunov, Cand. Sci. (Eng.), Assoc. Prof., Assoc. Prof., Computers and Systems Department; vegunov@mail.ru

Конфликт интересов/Conflict of interest.

Автор заявляет об отсутствии конфликта интересов/The author declare no conflict of interest.

Поступила в редакцию/Received 06.02.2025.

Одобрена после/рецензирования Reviced 30.03.2025.

Принята в печать/ Accepted for publication 17.05.2025.