УДК 519.87
DOI: 10.26102/2310-6018/2026.58.7.010

Двухмоментная оценка совместного исполнения независимых программных последовательностей

Е.А. Буевич

Московский государственный технологический университет «СТАНКИН», Москва, Российская Федерация

Резюме. Компиляция исходного кода на языке высокого уровня в машинный код является NP сложной задачей. Одной из проблем, возникающих в процессе компиляции и имеющих огромное пространство решений, является наличие в коде несвязанных по данным последовательностей операций. В данной работе рассматривается алгоритм поиска оптимального равномерного смешивания произвольных последовательностей инструкций, которые можно представить в виде ациклического связного графа. Целью подобного смешивания является увеличение параллелизма уровня инструкций. Для этого последовательность описывается как вектор случайных величин, каждая из которых характеризует его потребность в определенном ресурсе. Подобное представление позволяет приблизительно оценить математическое ожидание задержек, возникающих при исполнении их смеси, вследствие конкуренции нескольких последовательностей за ресурсы, без выполнения собственно компиляции. Алгоритм оперирует скалярными данными, описывающими последовательности и их комбинации, вместо длинных векторов инструкций. Поиск оптимального разбиения последовательностей на группы выполняется за время $O(n^k)$, где n – число последовательностей, а k – некоторое предельное значение числа последовательностей в смеси, зависящее от кода, латентности инструкций и числа регистров. Для вычислительных ядер современных процессоров в большинстве практических задач k не превышает 4.

Ключевые слова: параллелизм уровня инструкций, ациклический граф зависимостей, алгоритм Рейзера-Лавенберга, замкнутая сеть массового обслуживания, смешивание кода, оптимизация компилятора.

Для цитирования: Буевич Е.А. Двухмоментная оценка совместного исполнения независимых программных последовательностей. *Моделирование, оптимизация и информационные технологии.* 2026;14(7). URL: https://moitvivt.ru/ru/journal/article?id=2433 DOI: 10.26102/2310-6018/2026.58.7.010

Two-moment evaluation of joint execution of independent program sequences

E.A. Buevich

Moscow State Technological University "STANKIN", Moscow, the Russian Federation

Abstract. Compiling high-level language source code into machine code is an NP-hard problem. One of the problems arising during compilation, which has a huge solution space, is the presence of data-independent sequences of operations in the code. This paper considers an algorithm for finding an optimal uniform mixing of arbitrary instruction sequences, which can be represented as an acyclic connected graph. The purpose of such mixing is to increase the instruction-level parallelism. To this end, a sequence is described as a vector of random variables, each of which characterizes its demand for a specific resource. This representation allows for an approximate estimation of the expected value of delays arising during the execution of their mixture due to competition among several sequences for resources, without actually compiling the code. The algorithm operates on scalar data describing sequences and their combinations, instead of long instruction vectors. The search for the optimal

partitioning of sequences into groups is performed in $O(n^k)$ time, where n is the number of sequences, and k is some limiting value of the number of sequences in the mixture, depending on the code, instruction latency, and the number of registers. For computing cores of modern processors, in most practical problems k does not exceed 4.

Keywords: instruction-level parallelism, acyclic dependence graph, Reiser-Lavenberg algorithm, closed queueing network, code mixing, compiler optimization.

For citation: Buevich E.A. Two-moment evaluation of joint execution of independent program sequences. *Modeling, Optimization and Information Technology.* 2026;14(7). (In Russ.). URL: https://moitvivt.ru/ru/journal/article?id=2433 DOI: 10.26102/2310-6018/2026.58.7.010

Введение

Несвязные графы зависимостей порождают проблему поиска оптимального решения, имеющую в отсутствие оптимизаций экспоненциальный класс сложности [1]. Наиболее простым примером несвязанных графов являются циклы, итерации которых не связаны по данным, либо допускают редукцию. Данная задача является наиболее исследованным случаем и решается за константное или линейное время из-за того, что смешиваемый код одинаков и потребность в ресурсах растет пропорционально количеству совмещаемых итераций.

Более сложной является задача совмещения в одном потоке управления итераций двух разных циклов A и B, поскольку их потребности в ресурсах различны. Данный прием известен как Loop Fusion. В работе [2] предложен алгоритм поиска кандидатов для слияния циклов с различным числом итераций. В [3] дан обзор различных алгоритмов слияния на базе поиска с ограничениями или динамического программирования. Отмечено, что целью слияния является либо увеличение параллелизма, либо повышение локализации данных. Применительно к несвязным графам целью может быть только увеличение параллелизма уровня инструкций.

Однако любой ациклический граф может быть представлен как цикл с одной итерацией и таким образом стать кандидатом на слияние с другим ациклическим графом или его участком. Слиянию логически несвязанных участков кода уделяется меньше внимания, чем слиянию циклов, хотя подобную задачу часто удается сформулировать. Например, работа [4] посвящена ручному совмещению двух этапов выполнения операций линейной алгебры.

Часто эта проблема является следствием конвейеризации. Так, в [5] необходимость в подобной операции появилась как следствие сильной конвейеризации поиска в хеш-таблице. В результате разделения алгоритма на этапы получилось 6 независимых друг от друга по данным цепочек инструкций, четыре из которых являются циклами и, соответственно, также допускают разделение на несвязанные блоки. Результат, сгенерированный двумя актуальными компиляторами, был впоследствии вручную улучшен на ~20 % при помощи следующей эвристики. Каждому блоку была присвоена условная ширина – его потребность в абстрактном «ресурсе». Между блоками было введено отношение частичного порядка, основанное на времени их совместного выполнения, что позволило отсечь рассмотрение заведомо неоптимальных вариантов. Недостатки подобного подхода очевидны – поскольку блоки конкурируют за разные виды ресурсов, их взаимная несовместимость слабо отражает реальную картину.

В данной работе предлагается алгоритм, основанный на схожей эвристике, но учитывающий как различие ресурсов, так и нарастающее взаимное влияние компонентов по мере усложнения смеси, проявляющееся обычно в супераддитивности задержек. Ресурсы, за которые происходит конкуренция, можно разделить на два вида: вычислительные модули и единицы хранения (регистры). Ключевое различие между

ними заключается в том, что время занятости вычислительного модуля единичной операцией не зависит от кода, в то время как время занятости регистра зависит только от кода.

Материалы и методы

Моделированию микропроцессоров посвящен ряд работ. Например, в работе [6] достаточно детально воспроизведены многие составляющие его функционирования, но в контексте задачи смешивания потоков большую роль играет разделение инструкций по потокам и, в силу этого, их взаимная зависимость или независимость.

Для моделирования этого аспекта вычислительное ядро можно представить как закрытую сеть массового обслуживания. В качестве задания будет рассматриваться последовательность I, представляющая собой x_i последовательных зависимых по данным инструкций (применительно к современным процессорам под инструкцией понимается микрооперация). Каждая инструкция характеризуется некоторым набором вычислительных модулей V, на котором она может выполниться, некоторым временем занятости одного из этих модулей S и некоторой латентностью Z, после которой станет возможным исполнение следующей инструкции («временем размышлений» в терминологии СМО).

Тогда последовательность можно представить как набор случайных величин, описанных через моменты или массовую функцию. Обозначим Z_I – математическое ожидание латентности; ${c_{Z,I}}^2$ – квадрат коэффициента вариации латентности; $P_I(V)$ – вероятность обслуживания инструкции на наборе V, $\sum_V P_I(V) = 1$; $S_{I,V}$ – среднее время обслуживания инструкции на наборе V; ${c_{S,I,V}}^2$ – квадрат коэффициента вариации времени обслуживания на наборе V.

Определим операцию смешивания $A \oplus B \oplus \ldots$, обозначающую одновременное исполнение инструкций нескольких последовательностей. В рамках модели примем, что инструкции меньшей по времени исполнения последовательности равномерно распределяются по большему временному промежутку, что приводит к пропорциональному увеличению времени ожидания.

Для этого расширим понятие последовательности до блока вычислений, в котором может содержаться несколько таких последовательностей. Состав такого блока можно задать в виде бинарного вектора $\bar{n}$, в котором единицы соответствуют имеющимся в блоке последовательностям. Вектор, соответствующий единственному заданию I в блоке, обозначим как $\bar{e}_I$. Множество индексов, входящих в вектор $\bar{n}$, обозначим как $K(\bar{n})$.

Оценка задержек на ожидании вычислительных модулей. Для того, чтобы получить время ожидания каждой последовательности в блоке в очередях, используется модифицированный рекурсивный алгоритм Рейзера-Лавенберга, основанный на методе анализа средних значений (MVA) [7]. Поскольку число объединяемых последовательностей невелико, используя теорему о прибытии, можно последовательно рассчитать полный булеан всех возможных сочетаний заданий, начав с полного отсутствия заданий, далее рассмотрев все варианты с одним заданием (слой 1), далее рассмотрев все варианты с двумя заданиями, пользуясь характеристиками, полученными на предыдущем этапе и так далее.

Вычисления выполняются рекурсивно, добавлением еще одной последовательности к уже имеющимся, для которых нагрузка рассчитана на предыдущем шаге, начиная с пустой системы. Номер добавляемой последовательности больше, чем номера уже имеющихся. Длины очередей к приборам L_j и интенсивности λ_I для каждой заявки в этом состоянии равны нулю.

Для каждого состояния $\bar{n}$ вычисляются:

– скорректированное математическое ожидание латентности:

$$Z'_I = Z_I \cdot \frac{T_{min\{K(\bar{n})\}}(\bar{e}_{min\{K(\bar{n})\}})\cdot x_{min\{K(\bar{n})\}}}{T_I(\bar{e}_I)\cdot x_I}. \quad (1)$$

На первом слое принимается $Z'_I = Z_I$.

– динамический вес потока индивидуальной последовательности I на группе приборов V. Данной последовательности не было в системе на предыдущем шаге $(\bar{n} - \bar{e}_I)$, поэтому используется ее начальная интенсивность:

$$\lambda_{I,V,init} = P_I(V) \cdot \frac{1}{Z'_I + \sum_V P_I(V) S_{I,V}},$$

$$\alpha_{I,V}(\bar{n}) = \frac{\lambda_{I,V,init}}{\lambda_{I,V,init} + \sum_{m \in K(\bar{n}-\bar{e}_I)} \lambda_{m,V}(\bar{n}-\bar{e}_I)}. \quad (2)$$

Для первого слоя $\alpha_{I,V}(\bar{n}) = 1$, поскольку других заявок в системе нет.

– коэффициенты вариации потоков (аппроксимация Альпера [7]) для набора приборов V:

$$c_{in,V}{}^2(\bar{n}) = \left(1 - \rho_V(\bar{n})\right) \cdot B_Z + \rho_V(\bar{n}) \cdot B_S, \quad (3)$$

где $B_Z = \sum_{k \in K(\bar{n})} \alpha_{k,V}(\bar{n}) \left(P_k(V) c_{Z,k}{}^2 + 1 - P_k(V)\right)$ – вариация латентности и выбора набора, $B_S = \sum_{k \in K(\bar{n})} \alpha_{k,V}(\bar{n}) c_{S,k,V}{}^2$ – вариация обслуживания, $\rho_V(\bar{n}) = \sum_{k \in K(\bar{n})} \lambda_{k,V} \cdot S_{k,V}$ – загрузка прибора.

– дискретный мультипликатор Рейзера [8] для непуассоновских потоков:

$$\phi_V(\bar{n}) = \left(1 - \frac{1 - c_{in,V}{}^2(\bar{n})}{2 \sum_{m \in K(\bar{n})} \bar{e}_m}\right)\left(\frac{c_{in,V}{}^2(\bar{n}) + \sum_{k \in K(\bar{n})} \alpha_{k,V}(\bar{n}) \cdot c_{S,k,V}{}^2}{2}\right), \quad (4)$$

где $\sum_{m \in K(\bar{n})} \bar{e}_m$ – число последовательностей в блоке. Поскольку заявка приходит ровно на границе такта и может сразу же начать обслуживаться, поправка на ожидание окончания такта отсутствует.

– по закону Литтла среднее время цикла:

$$T_I(\bar{n}) = Z'_I + \sum_V P_I(V) \cdot R_I(V, \bar{n}). \quad (5)$$

Для получения $R_I(V, \bar{n})$ для каждого набора V каждой последовательности I находится время ожидания W на основании данных очередей к приборам из предыдущего шага:

$$W_I(V, \bar{n}) = S_{I,V} \cdot \phi_V(\bar{n}) \cdot \frac{\sum_{j \in V} L_j(\bar{n} - \bar{e}_I)}{\|V\|}, \quad (6)$$

где $\|V\|$ – количество приборов в наборе V, и соответственно время пребывания на наборе V для этой последовательности:

$$R_I(V, \bar{n}) = S_{I,V} + W_I(V, \bar{n}). \quad (7)$$

– из среднего времени цикла получается общая интенсивность:

$$\lambda_I(\bar{n}) = \frac{1}{T_I(\bar{n})}. \quad (8)$$

– интенсивность на набор приборов V и на конкретный прибор J:

$$\lambda_{I,V}(\bar{n}) = \lambda_I(\bar{n}) P_I(V), \quad (9)$$

$$\lambda_{I,J}(\bar{n}) = \lambda_I(\bar{n}) \cdot \sum_{V: j \in V} \frac{P_I(V)}{\|V\|}.$$

– длина очереди на каждом приборе для текущего состояния для следующего шага:

$$L_J(\bar{n}) = \sum_{k\in K(\bar{n})} \lambda_{k,J}(\bar{n}) \cdot R_k(V, \bar{n}). \quad (10)$$

После расчета первого слоя, для которого порядок неважен, последовательности сортируются по убыванию $T_I(\bar{e}_I) \cdot x_I$. Следствием такого порядка обхода будет константное значение Z'_I внутри блока при дальнейших расчетах.

Время выполнения блока целиком будет определяться самой длинной последовательностью в его составе.

$$D(\bar{n}) = max\{T_I(\bar{n}) \cdot x_I\}. \quad (11)$$

Операция смешивания в таком виде коммутативна $A \oplus B = B \oplus A$ и ассоциативна $(A \oplus B) \oplus C = A \oplus (B \oplus C)$, исходя из ее реализации. Данное свойство позволяет заменить размещения на сочетания при комбинаторном поиске.

Если в последовательностях нет сложных составных инструкций, то в силу того, что исполнительные порты имеют конвейерную организацию, можно принять $S_{I,V} = 1$, и также не нужно учитывать остаточное время обслуживания (поскольку оно завершается всегда на границе такта). Тогда уравнения (3) и (4) упрощаются:

$$c_{in,V}{}^2(\bar{n}) = 1 + \sum_{k\in K(\bar{n})} \alpha_{k,V}(\bar{n}) \cdot \left(c_{Z,k}{}^2 - 1\right), \quad (12)$$

$$\phi_V(\bar{n}) = \left(1 - \frac{1-c_{in,V}{}^2(\bar{n})}{2\sum_{m\in K(\bar{n})} \bar{e}_m}\right)\left(\frac{c_{in,V}{}^2(\bar{n})}{2}\right). \quad (13)$$

Важным условием применимости модели является свойство:

$$D(A \oplus B) \geq D(A). \quad (14)$$

Хотя оно очевидно для блоков, состоящих из одной последовательности, для составных блоков оно не является обязательным. Теоретически возможно существование последовательности-синхронизатора, которая снизит вероятность взаимных коллизий между другими последовательностями, увеличивая вариацию интервала между их последовательными обращениями к одному ресурсу [9]. Для такой последовательности, если она существует, должно выполняться неравенство

$$\sum_V \phi_V(\bar{n}) > \sum_V \phi_V(\bar{n} + \bar{e}). \quad (15)$$

Для практических последовательностей это маловероятно, поскольку в подавляющем большинстве случаев в силу небольшой разницы в латентности и времени обслуживания индивидуальных инструкций $c_{in,V}{}^2(\bar{n}) < 1$ и, таким образом, поток является субпуассоновскими. Из невозможности (15) следует, что добавление новых заявок ведет к монотонному росту длины очередей [10].

Для детекции подобной ситуации используется формула (14), поскольку $c_{in,V}{}^2(\bar{n}) > 1$, и как следствие формула (15) являются необходимыми, но не достаточными условиями.

Неравенство (14) важно как индикатор остановки смешивания. Если оно выполняется, то, при $D(A \oplus B) \geq D(A)+D(B)$ для всех вариантов слоя, дальнейшее добавление последовательностей в блок не приведет к оптимизации. Такой блок будем называть насыщенным, а входящие в его состав блоки – ненасыщенными. Это сокращает число возможных комбинаций последовательностей с 2^n до C_n^k, где k – максимальное число последовательностей в блоке, зависящее от кода, латентности инструкций и числа регистров, что позволяет перейти от экспоненциальной к степенной сложности алгоритма.

В случае, если при расчете была найдена последовательность-синхронизатор, для каждого насыщенного блока необходимо дополнительно проверить возможность смешивания с ней, и, если оно выгодно, отметить блок как ненасыщенный и продолжить поиск для него.

Оценка задержек, вызванных вытеснением данных другой последовательности из регистров. Задержки происходят из необходимости сохранять и перечитывать значения регистров. Использование регистров последовательностью I можно также описать через случайную величину Y_I, заданную массовой функцией y_I. Поскольку область ее определения невелика (в среднем не превышает число регистров), для блока последовательностей можно просто использовать свертку от компонентов.

$$y(\bar{n}) = \prod_{m \in K(\bar{n})} \cdot y_m. \tag{16}$$

Тогда математическое ожидание частоты превышения количества регистров R составит:

$$L_R(\bar{n}) = 1 - \sum_{m=1}^{R} y(\bar{n})(m). \tag{17}$$

Использование модулей в результате вытеснения можно задать в виде функции $L_I(V)$, описывающей количество использований каждого набора модулей V в результате единичного штрафа.

В отличие от использования ресурсов, в случае регистров важен порядок выполнения смешивания, поскольку он влияет на то, в какие именно исходные последовательности добавятся штрафные инструкции. Однако задание приоритета вытеснения позволит сохранить свойство ассоциативности, и в то же время учесть повышение нагрузки на соответствующие модули. Наиболее оптимальным выглядит вытеснение из регистров последовательности с минимальным числом операций, поскольку это минимизирует общую потерю производительности. На участке кода итогового блока, соответствующего превышению числа регистров, минимальное количество операций будет обращаться непосредственно к памяти. По этой причине исходные последовательности удобно пронумеровать в порядке возрастания их длины в инструкциях. Тогда величину $L_R(\bar{n})$ можно разделить между первыми исходными последовательностями в смеси так, чтобы штраф для последовательности I не превышал Y_I. Результатом будет вектор $\bar{l}_R$.

Его влияние на последовательность имеет два проявления – он увеличивает вероятность использования модулей загрузки и сохранения, и в то же время увеличивает длину последовательности. В зависимости от состава смеси на итоговое время выполнения повлиять могут как оба фактора, так и ни один из них, пока смесь недостаточно насыщена. Чтобы их учесть, до вычисления формулы (1) на каждом шаге модифицируем длину последовательностей и вероятности наборов:

$$x_i' = x_i + l_{R,i} \cdot \sum_V L_I(V), \tag{18}$$

$$P'_I(V) = \frac{P_I(V) \cdot x_i + L_I(V) \cdot l_{R,i}}{x_i'}.$$

Скорректированные значения $P'_I(V)$ используем вместо $P_I(V)$ на протяжении всего шага (формулы (2), (5), (9), (10)). Скорректированное значение x_i' используется в формулах (1) и (11).

После получения оценки для всех возможных насыщенных и ненасыщенных блоков оптимальное разбиение может быть найдено комбинаторным поиском с ограничениями, в котором рассматриваются только комбинации, не пересекающиеся с уже использованными. При этом есть два ограничения, позволяющие значительно

сократить число вариантов, оба являющиеся следствием выполняемости неравенства (14):

$$D(result) + D(outer_{\max}) < D(best), \tag{19}$$

где $D(result)$ – время выполнения уже добавленных блоков, $D(outer_{max})$ – время выполнения самой долгой последовательности, не входящей в добавленные, $D(best)$ – лучший найденный вариант. Нарушение неравенства (19) приводит к исключению вариантов, в которые входят все блоки $result$.

$$D(A \oplus B) < D(A) + D(B). \tag{20}$$

Это инвертированное условие остановки. Из него следует, что можно исключить рассмотрение вариантов, в которых два или более блоков входят в уже рассмотренный. Оптимально начинать поиск с набора наиболее крупных блоков, поскольку они исключают наибольшее количество вариантов, входящих в их состав.

Результаты и обсуждение

Исходный код реализации доступен по ссылке https://github.com/evgnort/merger/blob/main/mixer.c. Программа принимает на вход имена файлов в формате ассемблера AT&T и выводит лучшую найденную комбинацию.

Результат применения данного алгоритма для фрагментов кода из работы [5], (файлы samples/sampleA.s ... samples/sampleF.s, соответствующих стадиям конвейера в файле https://github.com/evgnort/hashtable/blob/main/pipeline_avx2.c) близок к варианту, найденному в результате ручной оптимизации.

Исходный набор блоков с флагами компиляции DREADABLE-DNO_MEM_WORK выглядит как

$$A + B + C + C + D + D + E + E + F + F.$$

Время его выполнения (gcc/msvc): 110/108 нс.

Соответствующие блоки размечены в исходном коде как 1,2,1,2,1,2 (A,B,C,D,E,F). Операция $+$ обозначает последовательное выполнение.

Оптимальный вариант после эвристической оптимизации, описанной в начале данной статьи (компиляция с флагом DNO_MEM_WORK):

$$(A \oplus B) + (C \oplus D \oplus E \oplus F) + (C \oplus D \oplus E \oplus F).$$

Время его выполнения (gcc/msvc): 92/89 нс.

Оптимальный вариант, предложенный программой mixer:

$$(A \oplus B) + (C \oplus C \oplus F \oplus F) + (D \oplus D \oplus E \oplus E).$$

Оценочное время его выполнения 143 нс.

Время поиска оптимального варианта: 0,319 с. Количество рассмотренных вариантов: 2736.

Алгоритм подтвердил, что слияние двух самых крупных последовательностей $A \oplus B$ дает максимальный выигрыш, а результат их слияния является насыщенным. Перераспределение итераций циклов в последующих блоках не является принципиальным, поскольку результат больше зависит от оптимизации в масштабе инструкций. Оба блока $(C \oplus C \oplus F \oplus F)$ и $(D \oplus D \oplus E \oplus E)$ не являются насыщенными, однако их равное разделение дает более быстрый результат, чем насыщение одного из них. Оценочное время завышено, поскольку компилятор располагает инструкции в более выгодном, чем случайный, порядке.

Интересным с точки зрения выполнимости условия (13) является наличие четырех инструкций vpgatherdd (латентность 22) в блоке A, что делает его коэффициент вариации латентности суперпуассоновским (1,13). Однако комбинации последовательностей, для которой бы нарушалось это условие, найдено не было, хотя из-за сравнительной легковесности кода с C по F, в некоторых ветках итогового варианта рассматривалось до 6 последовательностей в блоке.

Заключение

Поскольку в модели не учитываются оптимизации gcc, которые как применены к исходному варианту, так и применяются к результату (в оптимизированном случае пространства для них остается меньше), итоговый результат оказывается значительно быстрее, чем прогнозируется, исходя из представления блоков как случайной смеси инструкций. Автоматическое, без итерационной компиляции и исполнения, нахождение варианта, не худшего, чем результат длительной ручной оптимизации, подтверждает возможность использования описанного агрегированного представления кода. Алгоритм может быть использован компилятором при обнаружении нескольких несвязанных по данным цепочек инструкций внутри одного узла графа зависимостей по управлению для выбора стартовой точки поиска с ограничениями на базе отдельных инструкций.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Ахо А.В., Лам М.С., Сети Р. и др. *Компиляторы: принципы, технологии и инструментарий. 2-е изд.* Москва: ООО «И.Д. Вильямс»; 2017. 1184 с.
2. Логунов Б.А., Харин И.А. Методика разработки скоростного компилятора на основе модифицированного метода оптимизации loop fusion: модели и инструменты его реализации. *Computational Nanotechnology.* 2023;10(1):103–111. https://doi.org/10.33693/2313-223X-2023-10-1-103-111
   Logunov B.A., Kharin I.A. Methodology for Developing a High-speed Compiler Based on the Modified Loop Fusion Optimization Method: *Computational Nanotechnology.* 2023;10(1):103–111. (In Russ.) https://doi.org/10.33693/2313-223X-2023-10-1-103-111
3. Ziraksima M., Lotfi S., Izadkhah H. Loop Fusion Methods: A Critical Review. *International Journal of Advanced Research in Computer and Communication Engineering*. 2017;6(7):1–8.
4. Xu R.G., Van Zee F.G., van de Geijn R.A. *GEMMFIP: Unifying GEMM in BLIS.* arXiv. URL: https://doi.org/10.48550/arXiv.2302.08417 [Accessed 16th May 2026].
5. Буевич Е.А. Влияние размера образа ключа на производительность хеш-таблиц в современных архитектурах. *Моделирование, оптимизация и информационные технологии.* 2025;13(3). https://doi.org/10.26102/2310-6018/2025.50.3.014
   Buevich E.A. Impact of key representation size on hash tables performance in modern CPUs. *Modeling, Optimization and Information Technology.* 2025;13(3). (In Russ.). https://doi.org/10.26102/2310-6018/2025.50.3.014
6. Carroll S., Ling W.M. *A Queuing Model for CPU Functional Unit and Issue Queue Configuration.* arXiv. URL: https://doi.org/10.48550/arXiv.1807.08586 [Accessed 16th May 2026].
7. Вишневский В.М. *Теоретические основы проектирования компьютерных сетей.* Москва: Техносфера; 2003. 512 с.
8. Reiser M., Lavenberg S.S. Mean-Value Analysis of Closed Multichain Queuing Networks. *Journal of the ACM.* 1980;27(2):313–322. https://doi.org/10.1145/322186.322195

9. Reiser M. A Queueing Network Analysis of Computer Communication Networks with Window Flow Control. *IEEE Transactions on Communications.* 1979;27(8):1199–1209. https://doi.org/10.1109/TCOM.1979.1094531
10. Suri R. A Concept of Monotonicity and Its Characterization for Closed Queueing Networks. *Operations Research.* 1985;33(3):469–703. https://doi.org/10.1287/opre.33.3.606

ИНФОРМАЦИЯ ОБ АВТОРЕ / INFORMATION ABOUT THE AUTHOR

Буевич Евгений Андреевич, аспирант, Московский государственный технологический университет «СТАНКИН», Москва, Российская Федерация.
e-mail: gftcompmod@gmail.com

Evgeniy A. Buevich, Postgraduate, Moscow State Technological University "STANKIN", Moscow, the Russian Federation.

Статья поступила в редакцию 18.05.2026; одобрена после рецензирования 22.06.2026; принята к публикации 14.07.2026.

The article was submitted 18.05.2026; approved after reviewing xx.xx.2026; accepted for publication 14.07.2026.