УДК 004.421, 519.688 DOI: 10.14529/cmse260201

ОПИСАНИЕ И АНАЛИЗ СВОЙСТВ, ОСОБЕННОСТЕЙ И ХАРАКТЕРИСТИК АЛГОРИТМОВ В ОТКРЫТОЙ ЭНЦИКЛОПЕДИИ СВОЙСТВ АЛГОРИТМОВ ALGOWIKI*

© 2026 А.С. Антонов

Научно-иссследовательский вычислительный центр
Московского государственного университета имени М.В. Ломоносова
(119234 Москва, ул. Ленинские горы, д. 1, стр. 4)
E-mail: asa@parallel.ru

Поступила в редакцию: 16.03.2026

Исследование свойств алгоритмов является важнейшим этапом в процессе их эффективной реализации на высокопроизводительных вычислительных системах. В данной статье рассматриваются свойства, особенности и характеристики алгоритмов, которые могут быть полезны для проведения такого анализа. Рассмотрены необходимые предварительные шаги, которые нужно выполнить для приведения алгоритмов к виду, в котором их можно анализировать и сравнивать между собой. Выделяются аналитические характеристики алгоритмов, то есть те свойства, которые могут быть описаны в числовом или формульном виде, при этом акцент делается на свойствах, связанных с параллелизмом. Многие из рассматриваемых свойств именно для алгоритмов предлагаются впервые, в то же время являясь аналогичными свойствам, которые принято рассматривать для программных реализаций. Все рассматриваемые свойства иллюстрируются несколькими примерами для хорошо известных алгоритмов, таких как суммирование элементов вектора, скалярное произведение двух векторов, перемножение двух плотных квадратных матриц и метод Гивенса (вращений) QR-разложения квадратной матрицы. В дальнейшем на базе исследования рассмотренных свойств предполагается решать задачи суперкомпьютерного кодизайна по совместному анализу свойств алгоритмов, программных реализаций и суперкомпьютерных систем.

Ключевые слова: алгоритм, параллелизм, суперкомпьютер, кодизайн, вычислительная сложность, параллельная сложность, ресурс параллелизма, ускорение, граф информационных зависимостей, AlgoWiki.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Антонов А.С. Описание и анализ свойств, особенностей и характеристик алгоритмов в Открытой энциклопедии свойств алгоритмов AlgoWiki // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2026. Т. 15, № 2. С. 5–25. DOI: 10.14529/cmse260201.

Введение

Для описания структуры вычислительных алгоритмов могут применяться разные методы. Подробно они были рассмотрены в статье [1]. Наиболее часто на практике используются следующие методы: словесное описание, блок-схемы, описания на традиционных языках программирования или с использованием псевдокода.

Описывать свойства параллельных алгоритмов оказывается еще сложнее [2]. Часто свойства алгоритмов дают только в словесном описании, без четких определений и объяснения их сущности. Такие описания не могут использоваться для автоматического анализа. Для большей строгости описания информационной структуры алгоритма часто используют графовые модели, в частности, граф информационных зависимостей или граф алгоритма [3] (см. раздел 1.6). Полезно бывает визуализировать графы информационных

*Статья рекомендована к публикации Программным комитетом Всероссийской научной конференции с международным участием «Параллельные вычислительные технологии (ПаВТ) 2026».

зависимостей, для чего можно использовать, например, систему AlgoView [4, 5], требующую описания информационной структуры анализируемых алгоритмов на специальном языке Algolang [6].

Наиболее проработанное описание структуры алгоритмов используется в Открытой энциклопедии свойств алгоритмов AlgoWiki [7]. Предложенная в ней структура описания вычислительного алгоритма [8] включает следующие разделы:

1. Свойства и структура алгоритмов
   - 1.1 Общее описание алгоритма
   - 1.2 Математическое описание алгоритма
   - 1.3 Вычислительное ядро алгоритма
   - 1.4 Макроструктура алгоритма
   - 1.5 Схема реализации последовательного алгоритма
   - 1.6 Последовательная сложность алгоритма
   - 1.7 Информационный граф
   - 1.8 Ресурс параллелизма алгоритма
   - 1.9 Входные и выходные данные алгоритма
   - 1.10 Свойства алгоритма
2. Программная реализация алгоритма
   - 2.1 Особенности реализации последовательного алгоритма
   - 2.2 Возможные способы и особенности параллельной реализации алгоритма
   - 2.3 Результаты прогонов
   - 2.4 Выводы для классов архитектур
3. Литература

Описание алгоритмов по этой схеме включает множество полезных свойств и является весьма содержательным. Однако энциклопедия AlgoWiki реализуется в виде wiki-проекта [9] на базе движка MediaWiki [10]. Использование идеологии wiki-проектов дает большие возможности по привлечению авторов к работе по совместному наполнению контента сайта, но приводит к тому, что вся информация хранится в слабо структурированном текстовом виде, практически непригодном для проведения автоматического анализа.

Поэтому, когда встает задача автоматически анализировать свойства описанных в проекте алгоритмов, приходится частично или полностью отходить от идеологии wiki-проектов. Отдельные страницы энциклопедии AlgoWiki, например, основная страница классификации алгоритмов [11], реализующая хорошо известную схему описания предметной области в виде цепочек «задача–метод–алгоритм–реализация» [12, 13], уже реализованы без использования wiki-технологии [9].

Первый шаг по направлению к автоматическому исследованию свойств алгоритмов в рамках энциклопедии AlgoWiki был сделан, когда на страницах описаний алгоритмов стали делать выноску с несколькими базовыми свойствами. Пример выноски с базовыми свойствами метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] приведен на рис. 1.

На таких выносках стали размещать как наиболее важные свойства последовательных алгоритмов (последовательная сложность, объем входных и выходных данных), так и некоторые характеристики, связанные с параллелизмом (высота и ширина ярусно-параллельной формы). Так эти свойства стали более заметными, однако решением вышеописанной проблемы это не является, потому что данные выноски формируются на базе текстовых описаний

Метод Гивенса (вращений) QR-разложения квадратной матрицы	
Последовательный алгоритм	
Последовательная сложность	$2n^3$
Объём входных данных	n^2
Объём выходных данных	n^2
Параллельный алгоритм	
Высота ярусно-параллельной формы	$11n - 16$
Ширина ярусно-параллельной формы	$O(n^2)$

Рис. 1. Пример выноски с базовыми свойствами алгоритма

с использованием механизма шаблонов [15]. При этом сами данные по-прежнему являются плохо структурированными и не подходят для автоматического анализа.

Решением, подходящим для автоматического анализа, является ввод данных для подобных выносок путем заполнения полей специальных веб-форм. Лучше всего, если поля форм в этом случае заполняются путем выбора из заранее подобранных фиксированных вариантов. Чтобы это стало возможно, требуется строго описать те свойства, которые важны для описания характеристик алгоритмов и будут в дальнейшем использоваться для их автоматического анализа, чему и посвящена данная статья. Для вариантов числовых параметров, характеризующих эти свойства, нужно зафиксировать варианты наиболее часто используемых на практике их значений.

В разделе 1 данной статьи рассмотрены подходы к описанию и анализу свойств, особенностей и характеристик алгоритмов, позволяющие выделить алгоритмы, которые можно автоматически оценивать и сравнивать между собой на основе вводимых в разделе 2 характеристик.

1. Подходы к описанию и анализу свойств, особенностей и характеристик алгоритмов

1.1. Оценки в лучшем, в худшем и в среднем случае

Многие численные характеристики алгоритмов существенно зависят от заданных входных данных. Для одних их значений алгоритм может работать значительно дольше, чем для других. Поэтому анализ обычно «заключается в нахождении минимального (для оптимистов), максимального (для пессимистов), среднего (для специалистов по теории вероятностей) значения» [16] (здесь автор писал об оценке времени исполнения алгоритма). Для произвольных характеристик часто приводят оценки в лучшем, худшем и среднем случае.

Про получение оценок в лучшем, худшем и среднем случае пишут многие другие авторы книг по исследованию свойств алгоритмов [17–22]. Оценки в худшем случае позволяют гарантировать, что при отсутствии других важных действующих факторов значения изучаемых характеристик в реализации не окажутся хуже этих оценок. Оценки в лучшем случае показывают, к чему может приблизиться рассматриваемая характеристика на са-

мом удачном наборе входных данных. Но разница между худшим/лучшим случаем и тем, что получается на практике, иногда может оказаться весьма существенной. Оценки в среднем случае несут в себе некоторые предположения о значениях реальных входных данных. Однако получение оценок в среднем случае не всегда осмысленно и не всегда приводит к правильным результатам. Заметим также, что есть разные подходы к тому, что в конкретном случае считать средним.

1.2. Точные оценки и асимптотические оценки

Практически во всех книгах по анализу свойств алгоритмов (например, [16, 18, 20, 22–28]) для оценок ключевых характеристик используют не их абсолютные значения, а оценивают их через функцию от входных данных задачи, зачастую приводя ее асимптотику, чаще всего в формате «О большое».

Строгое определение этого понятия приводится, например, в книге [16]: «Запись $O(f(n))$ всегда обозначает следующее: существуют положительные константы M и n_0, такие, что величина x_n, представленная в виде $O(f(n))$, удовлетворяет условию $|x_n| \leq M|f(n)|$ для всех целых $n \geq n_0$».

«О большое» позволяет оценить порядок рассматриваемой величины, в некоторых источниках называется также «скорость роста (rate of growth)» или «порядок роста (order of growth)» [18, 21].

Использование семантики «О большого» во многих случаях дает возможность заметно упростить оценки характеристик алгоритмов, но оставить возможность их асимптотического сравнения. Например, автор страницы «Метод сдваивания Стоуна для решения двудиагональных СЛАУ» [29] в энциклопедии AlgoWiki приводит оценку последовательной сложности алгоритма как $3(n-1)(\lceil \log_2(n-1) \rceil + 2)$. Такая формула сложна как просто для восприятия, так и для сравнения с другими алгоритмами. Приведение оценки к семантике «О большого» даст оценку $n \log_2 n$, что уже значительно проще и может использоваться в дальнейшем анализе.

1.3. Выделение главного параметра

Вообще говоря, алгоритмы могут иметь несколько входных параметров. При построении формул для их характеристик получаются многопараметрические функции, которые анализировать значительно сложнее; также существенно усложняется сравнение этих характеристик для различных используемых алгоритмов. Во многих источниках анализируют только простые однопараметрические алгоритмы.

В классической книге [30] сказано: «Нам хотелось бы связать с каждой конкретной задачей некоторое число, называемое ее размером, которое выражало бы меру количества входных данных». В книге [27] это описано более детально: «Для простоты входные данные представляются параметром n. Этот параметр пропорционален величине обрабатываемого набора данных и может обозначать:

- размер массива или файла при сортировке или поиске;
- степень полинома;
- количество символов в строке;
- другую абстрактную меру объема рассматриваемой задачи».

Выделение главного параметра рассматривается, в частности, также в работе [22].

Некоторые многопараметрические алгоритмы, не теряя общности с точки зрения оценки их характеристик, можно свести к однопараметрическим вариантам. Например, во многих случаях, когда в алгоритмах используются прямоугольные матрицы размера $m \times n$ для анализа характеристик таких алгоритмов зачастую достаточно рассмотреть частный случай квадратной матрицы размера $n \times n$.

Сведение многопараметрических алгоритмов к однопараметрическим вариантам зачастую может оказаться неприемлемым, но анализ и сравнение многопараметрических функций, описывающих характеристики таких алгоритмов, является сложной задачей, к которой можно будет вернуться в будущем, если возникнет такая необходимость.

1.4. Выделение основного типа данных

Почти в любом алгоритме операции выполняются над данными, которые в конкретных реализациях описываются объектами каких-то типов, используемых в языке программирования. В одной программной реализации для разных объектов могут использоваться данные разных типов, но обычно можно выделить один основной тип данных, над которыми выполняется большинство наиболее важных операций. Далее все основные характеристики алгоритма оцениваются с учетом операций только над данными этого типа.

Так, общепринятой практикой является при решении задачи суммирования элементов вектора вещественных чисел учитывать при оценках характеристик алгоритма только операции над самими вещественными числами, хотя очевидно, что для реализации этого алгоритма на многих распространенных языках программирования потребуются дополнительные операции над вспомогательными переменными (чаще всего целочисленными) для организации цикла, в котором выполняется суммирование элементов вектора. Обычно считается, что подобными операциями можно пренебречь, к тому же этих операций нет в самом алгоритме, они появляются только в его программной реализации.

1.5. Выделение значимых операций

Шаги любого алгоритма могут состоять из самых разных операций (арифметических, логических, операций чтения/записи данных, пересылки данных и т.д.) Для построения простых оценок характеристик алгоритма нужно выделить из этих операций самые важные с точки зрения анализа свойств алгоритма [28]. В [22] сказано: «Первый шаг при анализе алгоритма состоит в определении абстрактных операций, на которых основан алгоритм, чтобы отделить анализ от реализации». Это важное замечание, потому что программные реализации алгоритма могут содержать новые операции, которых непосредственно алгоритм не предусматривает. Некоторые из таких операций (чтения/записи, пересылки данных и т.п.) могут играть решающую роль для эффективности выполнения программы, но не являются операциями самого реализуемого алгоритма, и поэтому не должны входить в оценки его характеристик. Задача совместного анализа алгоритма, программной реализации и целевой вычислительной системы с учетом характеристик всех этих трех сущностей является более сложной и решается в рамках проекта по решению задачи суперкомпьютерного кодизайна [31].

На практике при подсчете числовых характеристик алгоритмов обычно ограничиваются одним основным набором операций. Часто для простоты предполагают, что операции из одного набора примерно равноценны по времени выполнения. Во многих численных алгоритмах таким основным набором является подмножество реализуемых арифметических

операций. Характеристики не численных алгоритмов оцениваются с использованием тех типов операций, на которые они ориентированы. Так, например, в алгоритмах сортировки и в алгоритмах на графах основными элементарными операциями являются операции сравнения, перестановки и т.д.

Если алгоритм описан на основе макроопераций (и его можно назвать в этом случае макроалгоритмом), то подсчитывать его численные характеристики можно либо в числе этих макроопераций (но тогда его, вероятно, будет сложно сравнивать с другими алгоритмами), либо предварительно расшифровывая макрооперации в наборы элементарных операций.

1.6. Граф информационных зависимостей

Одним из основных подходов при анализе свойств алгоритмов, особенно связанных с параллелизмом, является построение и исследование *графа информационных зависимостей* (называемого также *графом алгоритма*) [3]. Это ориентированный ациклический граф, вершины которого соответствуют срабатываниям операций алгоритма, а дуги — информационным зависимостям между ними. Состав операций, соответствующих вершинам графа, может быть разным — от элементарных операций до макроопераций (в этом случае используют понятие *макрографа информационных зависимостей*).

Часто используемым представлением графа информационных зависимостей является его *ярусно-параллельная форма (ЯПФ)* (*parallel form*) — такое представление графа алгоритма, в котором:

- все вершины разбиты на перенумерованные подмножества ярусов;
- начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины;
- между вершинами, расположенными на одном ярусе, не может быть дуг.

Под *высотой ЯПФ* понимают число ее ярусов. *Ширина яруса* — это число вершин, расположенных на данном ярусе, а *ширина ЯПФ* — это максимальная ширина ярусов в ЯПФ.

Канонической ярусно-параллельной формой называется ЯПФ, высота которой на единицу больше длины критического пути, а все входные вершины расположены на первом ярусе. Для заданного графа информационных зависимостей его каноническая ЯПФ единственна [3].

Для построения графов информационных зависимостей и получения их канонической ярусно-параллельной формы можно использовать, например, систему AlgoView [4, 5]. На рис. 2 показан макрограф информационных зависимостей для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант), приведенный на странице описания данного алгоритма в энциклопедии AlgoWiki [14]. На графе ярко-оранжевым цветом помечены вершины, относящиеся к текущему ярусу канонической ярусно-параллельной формы, темно-зеленым цветом — вершины, относящиеся к предыдущим ярусам, светло-зеленым цветом — вершины, относящиеся к последующим ярусам.

2. Аналитические характеристики алгоритмов

Разные авторы выделяют разные характеристики алгоритмов. Так, Дональд Кнут (Donald Knuth) считает необходимыми такие их особенности, как конечность, определенность, ввод, вывод и эффективность [16].

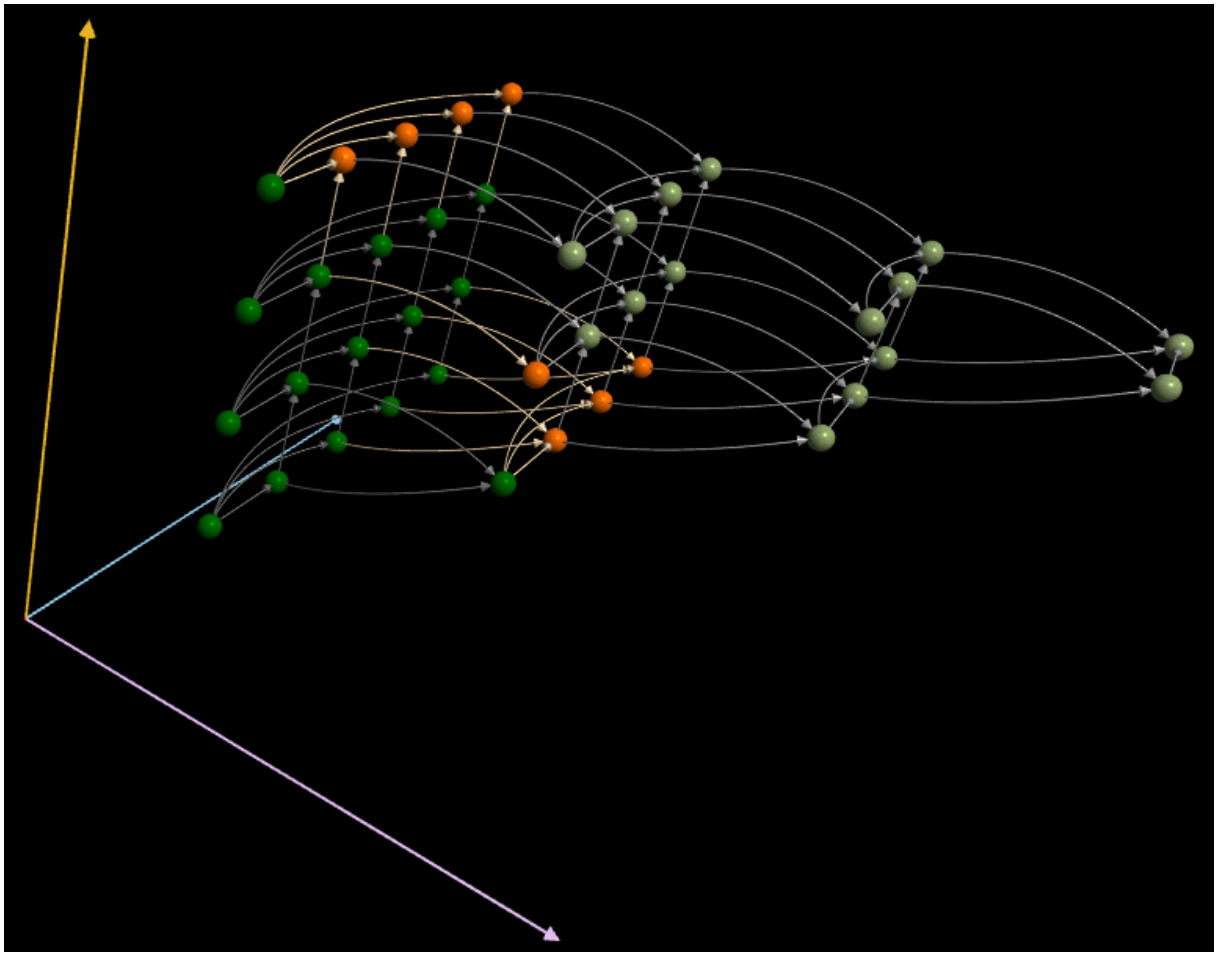

Рис. 2. Пример графа информационных зависимостей с выделением одного яруса канонической ЯПФ

В другом источнике [27] написано: «Алгоритмы должны обладать следующими формальными свойствами:

- понятность;
- дискретность;
- детерминированность;
- результативность;
- массовость».

В книге [18] сформулировано: «Хороший алгоритм должен обладать тремя свойствами: быть корректным, эффективным и легкореализуемым».

Таким образом, каждый автор выделяет свой набор ключевых характеристик алгоритмов, зачастую не давая строгих определений этим характеристикам. И даже в случае, если характеристика имеет четкое определение, она зачастую является описательной и не может быть просто охарактеризована числовым параметром.

Если же ставить цель автоматического или автоматизированного анализа свойств алгоритмов, то эти свойства должны быть четко описаны и измеримы. Основную идею анализа алгоритмов Д. Кнут [16] описывает как «взять конкретный алгоритм и определить его количественные характеристики». Целевой функцией такого анализа должна быть потенциальная возможность предсказания динамических характеристик алгоритма при его

реализации на целевой вычислительной системе: «Анализ алгоритма заключается в том, чтобы предсказать требуемые для его выполнения ресурсы» [21].

2.1. Детерминированность и безусловность

Важным свойством алгоритмов является их детерминированность. *Детерминированный алгоритм* — это алгоритм с постоянной структурой вычислительного процесса, который выдает уникальный и предопределенный результат для заданных входных данных. National Institute of Standards and Technology (NIST) в словаре Dictionary of Algorithms and Data Structures [32] определяет понятие детерминированный алгоритм как «Алгоритм, поведение которого можно полностью предсказать на основе входных данных (An algorithm whose behavior can be completely predicted from the input)». В книге [27] понятие детерминированности поясняется так: «Каждое правило алгоритма должно быть четким, однозначным и выдавать для одних исходных данных всегда один и тот же результат».

В книге [21] проводится четкое разделение между детерминированными и вероятностными (randomized) алгоритмами: «В общем случае алгоритм называется рандомизированным (randomized), если его поведение определяется не только набором входных величин, но и значениями, которые выдает генератор случайных чисел». То же самое разделение проводится и в книге [20]: «Алгоритмы, не использующие рандомизацию, называются детерминированными (Algorithms not using randomization are called deterministic)», а также в книге [18].

Однако не для всех детерминированных алгоритмов можно легко получить рассматриваемые в последующих пунктах характеристики. Так, итерационные алгоритмы (по определению зачастую являющиеся детерминированными) обычно требуют выполнения своих итераций до достижения определенной точности. Для одной такой итерации определение характеристик обычно возможно, но количество итераций заранее неизвестно и существенно зависит от входных данных. Для подобных алгоритмов получение подобных характеристик сильно затруднено и требует отдельного изучения.

В книге [3] дано следующее определение детерминированности: «Если программа не имеет условных операторов, то как ее саму, так и описанный ею алгоритм будем называть детерминированными. В противном случае будем называть их недетерминированными». Напрямую данное определение противоречит перечисленным выше классическим вариантам, но может быть основой для определения класса *безусловных* (*branchless*) алгоритмов. Итерационные алгоритмы не войдут в класс безусловных алгоритмов, поскольку для определения необходимости следующей итерации требуется проверка условия на достижение необходимой точности. Перечисленные в последующих пунктах характеристики рассматриваются для алгоритмов, принадлежащих к этому классу.

2.2. Вычислительная (последовательная) сложность

Под *вычислительной сложностью* (*computational complexity*) задачи W понимают количество основных вычислительных шагов лучшего последовательного алгоритма, необходимых для решения задачи на одном процессе. Поскольку дается характеристика последовательной версии алгоритма, то вычислительную сложность часто называют также *последовательной сложностью*.

Не всегда очевидно, что понимать под основными вычислительными шагами алгоритма. Это могут быть как выделенные значимые операции (см. раздел 1.5), так и более крупные

шаги, состоящие из множества элементарных операций. Набор используемых для оценки операций определяется как особенностями самого рассматриваемого алгоритма, так и теми целями, которые ставятся при его анализе.

Вычислительная сложность условно характеризует время, требуемое для последовательной реализации алгоритма, поэтому многие авторы книг по свойствам алгоритмов [17, 22, 23, 28, 33] говорят непосредственно об оценке времени исполнения. При этом надо понимать, что это условное время выражается не в секундах, а в количестве основных вычислительных шагов. Если для простоты делается допущение, что любой основной вычислительный шаг выполняется за единицу времени, то эти две величины совпадают. В книге [21] явно говорится «время работы алгоритма ... измеряется в количестве элементарных операций». В книгах [27, 30] используется термин временная сложность.

В ряде источников [24, 26, 34] вычислительную сложность соотносят с выполняемой работой (work), Д. Кнут [16] описывает ее как «ожидаемую скорость выполнения алгоритма». Иногда разделяют понятия трудоемкости алгоритма, под которой понимают точную оценку количества выполняемых операций, и сложность алгоритма, дающую асимптотическую (см. раздел 1.2) оценку [19].

Вычислительная сложность, как и многие другие рассматриваемые далее характеристики алгоритмов, обычно является некоторой функцией от размера входных данных рассматриваемого алгоритма, выделенного в качестве главного параметра (см. раздел 1.3).

Эту и последующие характеристики покажем на трех простых примерах: алгоритме суммирования элементов вектора длины n, алгоритме скалярного произведения двух векторов длины n и алгоритме перемножения двух плотных квадратных матриц размера $n \times n$. На рис. 3 показаны графы информационных зависимостей трех рассматриваемых алгоритмов. На графах квадратные (кубические) вершины соответствуют операциям перемножения двух чисел, а шарообразные вершины — операциям суммирования двух чисел.

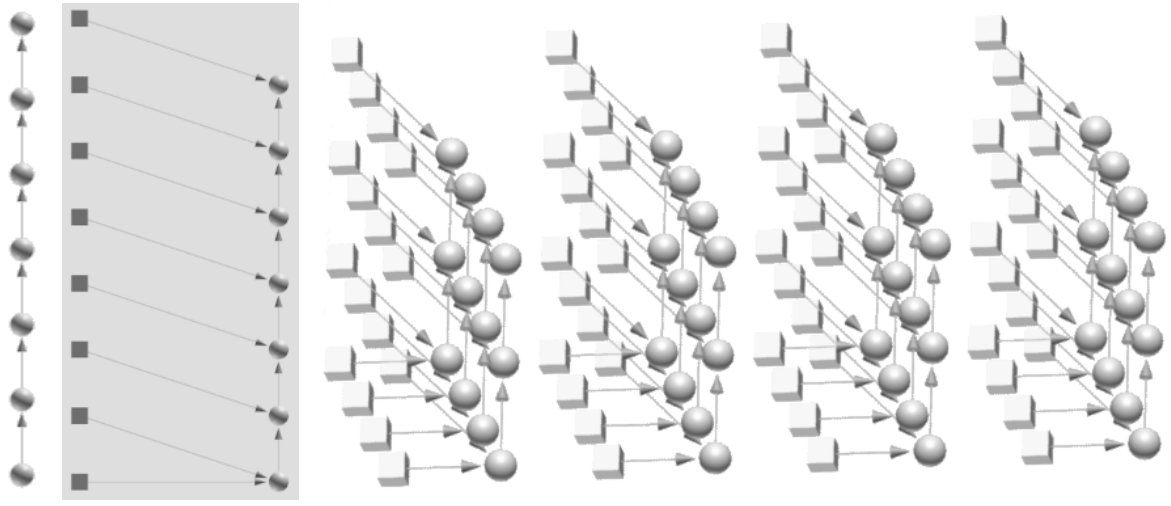

Рис. 3. Графы информационных зависимостей трех алгоритмов:
а) суммирования элементов вектора (для $n = 8$);
б) скалярного произведения двух векторов (для $n = 8$);
в) перемножения двух плотных квадратных матриц (для $n = 4$)

Вычислительная сложность алгоритма суммирования элементов вектора длины n, выраженная в количестве суммирований двух чисел, равна $W = n - 1$, вычислительная сложность алгоритма скалярного произведения двух векторов длины n при условии, что операции суммирования и перемножения двух чисел считаем равнозначными, составляет $W = 2n - 1$, а вычислительная сложность алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равна $W = n^2(2n - 1)$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] автор описания приводит оценку последовательной сложности как $W = 2n^3$, что после приведения к семантике «О большого» (см. раздел 1.2) можно было бы записать в виде $O(n^3)$.

2.3. Параллельная сложность

Под *параллельной сложностью* (*parallel complexity*) алгоритма W_p понимают количество основных вычислительных шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа необходимых вычислительных ресурсов (процессоров, функциональных устройств, вычислительных узлов, ядер и т.п.). Встречаются различные варианты названия этой характеристики: параллельная сложность (parallel complexity) [24], глубина вычислительной сети (depth of the computational network) [33], глубина (depth) [26], вычислительные шаги (computational steps) [17].

Параллельная сложность алгоритма соответствует высоте канонической ярусно-параллельной формы его графа информационных зависимостей (см. раздел 1.6), а также на единицу больше длины критического пути в данном графе, и характеризует минимально возможное число основных вычислительных шагов при параллельном исполнении рассматриваемого алгоритма.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Параллельная сложность алгоритма суммирования элементов вектора длины n равна последовательной сложности, т. е. $W_p = W = n - 1$ (алгоритм является строго последовательным; схема сдваивания и другие подобные методы изменяют информационную структуру, а значит, являются другими алгоритмами), параллельная сложность алгоритма скалярного произведения двух векторов длины n составляет $W_p = n$ (один шаг на все попарные произведения и $n-1$ шаг на суммирование), а параллельная сложность алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ тоже равна $W_p = n$ (все элементы результирующей матрицы можно вычислять параллельно, для вычисления каждого нужно выполнить скалярное произведение соответствующих строки и столбца исходных матриц).

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] автор описания приводит оценку параллельной сложности как $W_p = 11n - 16$, что после приведения к семантике «О большого» (см. раздел 1.2) можно было бы записать в виде $O(n)$.

2.4. Ресурс параллелизма

Одной из метрик *ресурса параллелизма* алгоритма можно считать ширину канонической ярусно-параллельной формы его графа информационных зависисимостей (см. раздел 1.6) W_r. Она характеризует максимальное число вершин графа информационных зависимостей, попавших на один ярус канонической ЯПФ. Такие вершины соответствуют

операциям, которые можно выполнять параллельно при наличии достаточного количества исполняющих устройств. В книге [17] соответствующее понятие называется числом процессоров (number of processors), при этом имеется в виду необходимое число процессоров для максимально параллельного выполнения алгоритма.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Ресурс параллелизма алгоритма суммирования элементов вектора длины n равен $W_r = 1$, ресурс параллелизма алгоритма скалярного произведения двух векторов длины n при условии, что операции суммирования и перемножения двух чисел считаем равнозначными, составляет $W_r = n$ (для попарных произведений), а ресурс параллелизма алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равен $W_r = n^3$ (все элементы результирующей матрицы можно вычислять параллельно, для вычисления каждого нужно выполнить скалярное произведение соответствующих строки и столбца исходных матриц).

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] автор описания приводит оценку ресурса параллелизма как $W_r = O(n^2)$.

2.5. Алгоритмическое ускорение

Понятие *ускорения* (*speedup*) чаще применяется как характеристика программных реализаций и определяется как $S = T_1/T_p$ — отношение времени работы некоторой программы на одном процессе T_1 ко времени работы распараллеленной программы при использовании p процессов T_p. Похожую характеристику можно рассмотреть и для алгоритмов. Поскольку вычислительная сложность (см. раздел 2.2) является некоторым приближением времени выполнения последовательного алгоритма, а параллельная сложность (см. раздел 2.3) — некоторым приближением времени выполнения параллельного алгоритма, то их отношение [28] $S_{max} = W/W_p$ будет аналогом ускорения для алгоритмов. В данном случае S_{max} определяет максимальное *алгоритмическое ускорение*, которое теоретически можно получить для реализации данного алгоритма в отсутствие влияния других значимых факторов при наличии бесконечного количества исполняющих устройств (в концепции неограниченного параллелизма [3]).

В книге [26] аналогичную характеристику называют просто параллелизм (parallelism), а в книге [17] — ускорение (speedup).

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Алгоритмическое ускорение суммирования элементов вектора длины n равно $S_{max} = (n-1)/(n-1) = 1$, алгоритмическое ускорение скалярного произведения двух векторов длины n составляет $S_{max} = (2n-1)/n$, а алгоритмическое ускорение перемножения двух плотных квадратных матриц размера $n \times n$ равно $S_{max} = n^2(2n-1)/n = 2n^2 - n$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] алгоритмическое ускорение можно оценить как $S_{max} = O(n^3)/O(n) = O(n^2)$.

2.6. Алгоритмическая эффективность распараллеливания

Понятие *эффективность распараллеливания* (*parallel efficiency*) чаще применяется как характеристика программных реализаций и определяется как $E_p = S/p$ (где S — полученное ускорение работы программы, а p — число используемых процессов. Похожую характе-

ристику можно рассмотреть и для алгоритмов. Поскольку алгоритмическое ускорение (см. раздел 2.5) S_{max} является некоторым приближением ускорения для параллельного алгоритма, а ресурс параллелизма (см. раздел 2.4) W_r — некоторым приближением необходимого количества процессов, то их отношение $E_{p_{max}} = S_{max}/W_r$ будет аналогом эффективности распараллеливания для алгоритмов. В данном случае $E_{p_{max}}$ определяет *алгоритмическую эффективность распараллеливания*, которая показывает, насколько хорошо может быть распараллелен данный алгоритм.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Алгоритмическая эффективность распараллеливания суммирования элементов вектора длины n равна $E_{p_{max}} = 1$ (в этом вырожденном случае высокая алгоритмическая эффективность распараллеливания не означает наличия параллельной реализации), алгоритмическая эффективность распараллеливания скалярного произведения двух векторов длины n составляет $E_{p_{max}} = \frac{2n-1}{n^2}$, а алгоритмическая эффективность распараллеливания алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равна $E_{p_{max}} = \frac{2n^2-n}{n^3} = \frac{2n-1}{n^2}$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] алгоритмическую эффективность распараллеливания можно оценить как $E_{p_{max}} = O(n^2)/O(n^2) = O(1)$.

2.7. Алгоритмическая стоимость

Понятие *стоимости* (*cost*) чаще применяется как характеристика программных реализаций и определяется как $C = pT_p$ — произведение количества процессов p на время работы распараллеленной программы T_p. Похожую характеристику можно рассмотреть и для алгоритмов. Поскольку параллельная сложность (см. раздел 2.3) W_p является некоторым приближением времени выполнения параллельного алгоритма, а ресурс параллелизма (см. раздел 2.4) W_r — некоторым приближением необходимого количества процессов, то их произведение $C_{max} = W_r * W_p$ будет аналогом стоимости для алгоритмов. В данном случае C_{max} задает *алгоритмическую стоимость*, которая определяется как суммарное количество операций, которые выполнились бы в параллельной реализации алгоритма при условии максимальной загрузки всех процессов. Оценки алгоритмической стоимости зачастую могут оказаться сильно завышенными, поскольку могут включать учет операций, которых нет в изначальном алгоритме.

Похожая характеристика стоимости (cost) рассматривается в ряде источников [17, 24, 26, 28, 34], а в книге [33] она называется «объем работы, выполняемой параллельным алгоритмом (the amount of work performed by a parallel algorithm)».

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Алгоритмическая стоимость суммирования элементов вектора длины n равна $C_{max} = n - 1$, алгоритмическая стоимость скалярного произведения двух векторов длины n составляет $C_{max} = n^2$, а алгоритмическая стоимость алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равна $C_{max} = n^4$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] алгоритмическую стоимость можно оценить как $C_{max} = O(n^3)$.

2.8. Алгоритмические накладные расходы

Понятие *накладных расходов* (*total overhead*) чаще применяется как характеристика программных реализаций и определяется как $T_0 = C - T_1$ — разница стоимости и времени работы программы на одном процессе T_1. Похожую характеристику можно рассмотреть и для алгоритмов. Поскольку алгоритмическая стоимость (см. раздел 2.7) C_{max} является некоторым аналогом стоимости для алгоритмов, а вычислительная сложность (см. раздел 2.2) W является некоторым приближением времени выполнения последовательного алгоритма, то их разность $T_{0max} = C_{max} - W$ будет аналогом накладных расходов для алгоритмов. В данном случае T_{0max} определяет *алгоритмические накладные расходы*, то есть оценку того, насколько суммарное количество операций параллельного алгоритма при условии максимальной загрузки всех процессов больше количества операций последовательного алгоритма.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Алгоритмические накладные расходы алгоритма суммирования элементов вектора длины n равны $T_{0max} = 0$ (что не удивительно для последовательного алгоритма), алгоритмические накладные расходы алгоритма скалярного произведения двух векторов длины n составляют $T_{0max} = n^2 - (2n - 1)$, а алгоритмические накладные расходы алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равны $T_{0max} = n^4 - n^2(2n - 1)$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] алгоритмические накладные расходы можно оценить как $T_{0max} = O(n^3) - O(n^3) = O(n^3)$.

2.9. Алгоритмическая избыточность

Иногда бывает более показательно сравнивать алгоритмическую стоимость (см. раздел 2.7) и вычислительную сложность (см. раздел 2.2) рассмотрением не разности, а отношения данных величин. При этом получается *алгоритмическая избыточность* (*redundancy*) $R = C_{max}/W$.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Алгоритмическая избыточность суммирования элементов вектора длины n равна $R = 1$, алгоритмическая избыточность скалярного произведения двух векторов длины n составляют $R = \frac{n^2}{2n-1} = O(n)$, а алгоритмическая избыточность перемножения двух плотных квадратных матриц размера $n \times n$ равна $R = \frac{n^4}{n^2(2n-1)} = O(n)$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] алгоритмическую избыточность можно оценить как $R = O(n^3)/O(n^3) = O(1)$.

2.10. Оценка необходимого объема памяти

Важным при анализе алгоритмов является не только оценка выполняемых операций, но и оценка *необходимого объема памяти*. Поскольку объем данных в единицах измерения информации (битах, байтах) определяется реализацией, то для алгоритмов необходимый объем памяти оценивают в количестве элементов основного типа данных (см. раздел 1.4). Эту характеристику называют также емкостная сложность [30], сложность по памяти [28] или пространственная сложность [27].

Оценка необходимого объема памяти состоит из двух частей: оценки объема входных данных [21] алгоритма V_{in} и оценки объема его выходных данных V_{out}.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Объем входных данных алгоритма суммирования элементов вектора длины n равен $V_{in} = n$, а объем выходных данных этого алгоритма $V_{out} = 1$, объем входных данных алгоритма скалярного произведения двух векторов длины n составляет $V_{in} = 2n$, а объем выходных данных этого алгоритма $V_{out} = 1$, объем входных данных алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равен $V_{in} = 2n^2$, а объем выходных данных этого алгоритма $V_{out} = n^2$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] объем входных и выходных данных можно оценить как $V_{in} = V_{out} = n^2$.

2.11. Вычислительная мощность

Для программных реализаций хорошо известна такая характеристика, как арифметическая интенсивность (arithmetic intensity [35]). Она соотносит интенсивность вычислений в программе с интенсивностью работы с оперативной памятью и определяется как отношение количества операций, выполняемых в программе, к количеству обращений к памяти. Для алгоритмов оценку количества выполняемых операций дает вычислительная сложность 2.2, а количество обращений к памяти в общем случае оценить достаточно сложно. Поэтому как характеристику алгоритмов рассматривают *вычислительную мощность* (*computational power*) — отношение вычислительной сложности (см. раздел 2.2) алгоритма к суммарному объему его входных и выходных данных (см. раздел 2.10) $P = \frac{W}{V_{in}+V_{out}}$.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Вычислительная мощность алгоритма суммирования элементов вектора длины n равна $P = \frac{n-1}{n+1}$, вычислительная мощность алгоритма скалярного произведения двух векторов длины n составляет $P = \frac{2n-1}{2n+1}$, а вычислительная мощность алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равна $P = \frac{n^2(2n-1)}{3n^2} = \frac{2n-1}{3}$.

Для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] вычислительную мощность можно оценить как $P = \frac{2n^3}{2n^2} = n$.

2.12. Свойства графа информационных зависимостей

Важные свойства алгоритмов можно получать, анализируя свойства соответствующих графов информационных зависимостей (см. раздел 1.6). Часть таких свойств однозначно соответствует описанным в предыдущих пунктах характеристикам:

- *Количество вершин* графа информационных зависимостей соответствует вычислительной сложности (см. раздел 2.2) алгоритма;
- *Высота канонической ярусно-параллельной формы* графа информационных зависимостей соответствует параллельной сложности (см. раздел 2.3) алгоритма;
- *Ширина канонической ярусно-параллельной формы* графа информационных зависимостей характеризует ресурс параллелизма (см. раздел 2.4) алгоритма.

Другие свойства графов информационных зависимостей дают дополнительные полезные характеристики соответствующих алгоритмов.

2.12.1. Количество дуг в графе информационных зависимостей

Количество дуг E в графе информационных зависимостей является одной из базовых характеристик графа, а применительно к анализу алгоритмов характеризует общее количество информационных зависимостей, что в конечном счете означает число обращений в память и в какой-то степени необходимых пересылок данных при параллельной реализации.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Количество дуг в графе информационных зависимостей алгоритма суммирования элементов вектора длины n равно $E = n-2$, количество дуг в графе информационных зависимостей алгоритма скалярного произведения двух векторов длины n составляет $E = 2(n-1) = 2n-2$, а количество дуг в графе информационных зависимостей алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равно $E = n^2(2n-2)$.

2.12.2. Плотность графа информационных зависимостей

Плотностью графа (*graph density*) принято называть соотношение количества дуг и вершин $D = E/W$. Иногда эту величину называют средней степенью вершины. Применительно к графам информационных зависимостей плотность характеризует интенсивность работы с данными в соответствующем алгоритме.

Рассмотрим примеры алгоритмов с графами информационных зависимостей, приведенными на рис. 3. Плотность графа информационных зависимостей алгоритма суммирования элементов вектора длины n равна $D = \frac{n-2}{n-1}$, плотность графа информационных зависимостей алгоритма скалярного произведения двух векторов длины n составляет $D = \frac{2n-2}{2n-1}$, а плотность графа информационных зависимостей алгоритма перемножения двух плотных квадратных матриц размера $n \times n$ равна $D = \frac{n^2(2n-2)}{n^2(2n-1)} = \frac{2n-2}{2n-1}$.

2.12.3. Количество типов дуг в графе информационных зависимостей

Типы дуг в графе информационных зависимостей задают разные образцы обращения к данным в алгоритме: считаем, что тип дуг определяется единым направляющим вектором и равной длиной при фиксированных значениях параметров. Поскольку направляющие векторы и длины дуг существенно зависят от расположения вершин, то определять разные типы дуг можно при некотором фиксированном (стандартном) способе их расположения.

На рис. 3 видно, что граф информационных зависимостей алгоритма суммирования элементов вектора содержит дуги только одного типа, а графы информационных зависимостей алгоритма скалярного произведения двух векторов и алгоритма перемножения двух плотных квадратных матриц содержат дуги трех типов.

2.12.4. Наличие в графе информационных зависимостей вершин со степенью исхода, зависящей от внешних параметров

Степень исхода (*degree of a vertex*) вершины графа информационных зависимостей — это количество исходящих из данной вершины информационных дуг. Если в графе информационных зависимостей есть вершины, степень исхода которых зависит от внешних параметров, то в программной реализации это во многих случаях будет соответствовать использованию массовых рассылок с помощью, например, коллективных операций в техно-

логии MPI. Такая информация будет полезной при написании программы, но может быть зафиксирована уже в момент анализа структуры алгоритма.

Графы информационных зависимостей алгоритмов на рис. 3 не содержат таких вершин, а в макрографе информационных зависимостей для метода Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) [14] на рис. 2 такие вершины присутствуют.

Заключение

В данной статье предложен набор свойств, особенностей и характеристик алгоритмов, предназначенных для обеспечения возможности их автоматического анализа. Следующим шагом планируется внесение этих свойств в описания алгоритмов на страницах Открытой энциклопедии свойств алгоритмов AlgoiWiki [7]. Для дальнейшего их автоматического анализа данные свойства будут вноситься авторами статей не в wiki-формате, а путем заполнения полей специальной веб-формы с занесением в базу данных проекта. В более отдаленной перспективе планируется разработка технологий автоматического получения таких свойств по некоторому описанию информационной структуры алгоритма (например, на языке Algolang [6]) или по тексту его программной реализации на традиционном языке программирования. И далее полученные свойства будут использоваться для решения задач суперкомпьютерного кодизайна [31] по совместному анализу свойств алгоритмов, программных реализаций и суперкомпьютерных систем.

Исследование выполнено за счет гранта Российского научного фонда № 25–11–00181, `https://rscf.ru/project/25-11-00181/`. *Работа выполнена с использованием оборудования Центра коллективного пользования сверхвысокопроизводительными вычислительными ресурсами МГУ имени М. В. Ломоносова [36].*

Литература

1. Антонов А. Обзор методов описания структуры алгоритмов // Вычислительные методы и программирование. 2025. Т. 26, № 4. С. 548–568. DOI: 10.26089/NumMet.v26r436.
2. Voevodin V., Antonov A., Dongarra J. Why is it hard to describe properties of algorithms? // Procedia Computer Science. 2016. Vol. 101. P. 4–7. DOI: 10.1016/j.procs.2016.11.002.
3. Воеводин В.В., Воеводин В.В. Параллельные вычисления. СПб: БХВ-Петербург, 2002. 608 с.
4. Antonov A., Volkov N. Information Graph Visualization Using AlgoView Software Tool // Lobachevskii J. Math. 2020. Vol. 41, no. 6. P. 1427–1434. DOI: 10.1134/S199508022008003X.
5. Skryabin G., Gadieva T., Antonov A. A New Version of the AlgoView System for 3D Visualization and Interactive Analysis of Information Graphs of Algorithms // Parallel Computational Technologies. Vol. 2241 / ed. by L. Sokolinsky, M. Zymbler, V. Voevodin, J. Dongarra. Cham: Springer, 2024. P. 19–33. Communications in Computer and Information Science. DOI: 10.1007/978-3-031-73372-7_2.
6. Antonov A., Volkov N. Study of the Algorithms Information Structure as the Basis of a Training Workshop // Supercomputing. Vol. 1510 / ed. by V. Voevodin, S. Sobolev.

Cham: Springer, 2021. P. 404–414. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-92864-3_31.

7. Открытая энциклопедия свойств алгоритмов. URL: https://algowiki-project.org (дата обращения: 23.01.2026).
8. Структура описания свойств алгоритмов — Алговики. URL: https://algowiki-project.org/en/Description_of_algorithm_properties_and_structure (дата обращения: 23.01.2026).
9. Antonov A. Wiki Representation and Analysis of Knowledge About Algorithms // Supercomputing. Vol. 13708 / ed. by V. Voevodin, S. Sobolev, M. Yakobovskiy, R. Shagaliev. Cham: Springer, 2022. P. 604–616. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-22941-1_44.
10. MediaWiki. URL: https://www.mediawiki.org (дата обращения: 23.01.2026).
11. Классификация алгоритмов — Алговики. URL: https://algowiki-project.org/en/Algorithm_classification (дата обращения: 23.01.2026).
12. Antonov A., Frolov A., Konshin I., Voevodin V. Hierarchical Domain Representation in the AlgoWiki Encyclopedia: From Problems to Implementations // Parallel Computational Technologies. Vol. 910 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2018. P. 3–15. Communications in Computer and Information Science. DOI: 10.1007/978-3-319-99673-8_1.
13. Popov A., Nikitenko D., Antonov A., Voevodin V. Formal Model of Problems, Methods, Algorithms and Implementations in the Advancing AlgoWiki Open Encyclopedia // CEUR Workshop Proc. Vol. 2281. 2018. P. 1–11. URL: http://ceur-ws.org/Vol-2281/#paper-01.
14. Метод Гивенса (вращений) QR-разложения квадратной матрицы (вещественный точечный вариант) — Алговики. URL: https://algowiki-project.org/en/Givens_method (дата обращения: 23.01.2026).
15. Справка: Шаблоны — MediaWiki. URL: https://www.mediawiki.org/wiki/Help:Templates/ru (дата обращения: 23.01.2026).
16. Кнут Д. Искусство программирования. Том 1. Основные алгоритмы. 3-е издание. М.: ООО «И.Д. Вильямс», 2001. 720 с.
17. Akl S. The Design and Analysis of Parallel Algorithms. Prentice Hall, 1989. 412 p.
18. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд.: Пер. с англ. СПб: БХВ-Петербург, 2011. 720 с.
19. Лобанова В., Воронина О., Лобанова Н. Теория алгоритмов: учебное пособие. Орёл: ОГУ имени И.С. Тургенева, 2017. 95 с.
20. Mehlhorn K., Sanders P. Algorithms and Data Structures: The Basic Toolbox. Springer Science & Business Media, 2008. 305 p. DOI: 10.1007/978-3-540-77978-0.
21. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ, 2-е издание. Пер. с англ. М.: Издательский дом «Вильямс», 2011. 1296 с.
22. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. К.: Издательство «ДиаСофт», 2001. 688 с.

23. Reingold E. Basic Techniques for Design and Analysis of Algorithms. Chapman, Hall/CRC, 2004. 3–1 p. DOI: 10.1201/9780429171529-4.

24. JaJa J. An Introduction to Parallel Algorithms. Addison Wesley Longman Publishing Co., Inc., United States, 1992. 566 p.

25. Alsuwaiyel M. Parallel algorithms. World Scientific Publishers, 2022. 400 p. DOI: 10.1142/12744.

26. Blelloch G., Maggs B. Parallel Algorithms. Computer science handbook / editor-in-chief, Allen B. Tucker. 2nd ed. 2004. 10-10–41 p.

27. Селиванова И., Блинов В. Фундаментальные алгоритмы на C++. Построение и анализ алгоритмов обработки данных: учебно-методическое пособие. Екатеринбург: Издательство Уральского университета, 2015. 108 с.

28. Макконнелл Д. Основы современных алгоритмов. 2-е дополненное издание. М.: Техносфера, 2004. 368 с.

29. Метод сдваивания Стоуна для решения двудиагональных СЛАУ — Алговики. URL: https://algowiki-project.org/en/Stone_doubling_algorithm_for_solving_bidiagonal_SLAEs (дата обращения: 23.01.2026).

30. Ахо А., Хопкрофт Д., Ульман Д. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.

31. Antonov A., Maier R., Nikitenko D., Voevodin V. An Approach to Solving the Problem of Supercomputer Co-design // Lobachevskii J Math. 2024. Vol. 45, no. 7. P. 2965–2973. DOI: 10.1134/S1995080224603680.

32. Deterministic algorithm. URL: https://xlinux.nist.gov/dads/HTML/deterministicAlgorithm.html (дата обращения: 23.01.2026).

33. Smith J. The Design and Analysis of Parallel Algorithms. Oxford University Press, 1993. 470 p.

34. Casanova H., Legrand A., Robert Y. Parallel Algorithms. CRC PRESS, 2020. 335 p.

35. Паттерсон Д., Хеннеси Д. Архитектура компьютеров и проектирование компьютерных систем. 4-е. СПб: Питер, 2012. 919 с. «Классика computer science».

36. Voevodin V., Antonov A., Nikitenko D., *et al.* Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community // Supercomputing Frontiers and Innovations. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.

Антонов Александр Сергеевич, к.ф.-м.н., ведущий научный сотрудник, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

DOI: 10.14529/cmse260201

DESCRIBING AND ANALYZING PROPERTIES, FEATURES AND CHARACTERISTICS OF ALGORITHMS IN THE ALGOWIKI OPEN ENCYCLOPEDIA OF PARALLEL ALGORITHMIC FEATURES*

© 2026 A.S. Antonov

Research Computing Center, Lomonosov Moscow State University
(GSP-1, Leninskie Gory 1, building 4, Moscow, 119234 Russia)
E-mail: asa@parallel.ru

Received: 16.03.2026

Investigating the properties of algorithms is a crucial step in their effective implementation on high-performance computing systems. This paper examines the properties, features, and characteristics of algorithms that can be useful for such analysis. It also discusses the necessary preliminary steps to transform algorithms into a form suitable for analysis and comparison. Analytical characteristics of algorithms – that is, those properties that can be described numerically or formulaically – are highlighted, with an emphasis on properties related to parallelism. Many of the properties discussed are proposed for algorithms for the first time, while at the same time being similar to those commonly considered for software implementations. All of the properties discussed are illustrated with several examples for well-known algorithms, such as summation of vector elements, the scalar product of two vectors, the multiplication of two dense square matrices, and the Givens (rotation) method of the QR factorization of a square matrix. In the future, based on the study of the considered properties, it is proposed to solve problems of supercomputer co-design for the joint analysis of the properties of algorithms, software implementations and supercomputer systems.

Keywords: algorithm, parallelism, supercomputer, co-design, computational complexity, parallel complexity, parallelism resource, speedup, information dependency graph, AlgoWiki.

FOR CITATION

Antonov A.S. Describing and Analyzing Properties, Features and Characteristics of Algorithms in the AlgoWiki Open Encyclopedia of Parallel Algorithmic Features. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2026. Vol. 15, no. 2. P. 5–25. (in Russian) DOI: 10.14529/cmse260201.

References

1. Antonov A. Review of methods for describing the structure of algorithms. Numerical Methods and Programming. 2025. Vol. 26, no. 4. P. 548–568. (in Russian) DOI: 10.26089/NumMet.v26r436.
2. Voevodin V., Antonov A., Dongarra J. Why is it hard to describe properties of algorithms?. Procedia Computer Science. 2016. Vol. 101. P. 4–7. DOI: 10.1016/j.procs.2016.11.002.
3. Voevodin V., Voevodin V. Parallel computing. BHV-Peterburg, 2002. 608 p. (in Russian).

*The article was recommended for publication by the program committee of the international scientific conference "Parallel Computing Technologies (PaVT) 2026".

4. Antonov A., Volkov N. Information Graph Visualization Using AlgoView Software Tool. Lobachevskii J. Math. 2020. Vol. 41, no. 6. P. 1427–1434. DOI: 10.1134/S199508022008003X.

5. Skryabin G., Gadieva T., Antonov A. A New Version of the AlgoView System for 3D Visualization and Interactive Analysis of Information Graphs of Algorithms. Parallel Computational Technologies. Vol. 2241 / ed. by L. Sokolinsky, M. Zymbler, V. Voevodin, J. Dongarra. Cham: Springer, 2024. P. 19–33. Communications in Computer and Information Science. DOI: 10.1007/978-3-031-73372-7_2.

6. Antonov A., Volkov N. Study of the Algorithms Information Structure as the Basis of a Training Workshop. Supercomputing. Vol. 1510 / ed. by V. Voevodin, S. Sobolev. Cham: Springer, 2021. P. 404–414. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-92864-3_31.

7. Open Encyclopedia of Parallel Algorithmic Features – Algowiki. URL: https://algowiki-project.org/en (accessed: 23.01.2026).

8. Description of algorithm properties and structure – Algowiki. URL: https://algowiki-project.org/en/Description_of_algorithm_properties_and_structure (accessed: 23.01.2026).

9. Antonov A. Wiki Representation and Analysis of Knowledge About Algorithms. Supercomputing. Vol. 13708 / ed. by V. Voevodin, S. Sobolev, M. Yakobovskiy, R. Shagaliev. Cham: Springer, 2022. P. 604–616. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-22941-1_44.

10. MediaWiki. URL: https://www.mediawiki.org (accessed: 23.01.2026).

11. Algorithm classification – Algowiki. URL: https://algowiki-project.org/en/Algorithm_classification (accessed: 23.01.2026).

12. Antonov A., Frolov A., Konshin I., Voevodin V. Hierarchical Domain Representation in the AlgoWiki Encyclopedia: From Problems to Implementations. Parallel Computational Technologies. Vol. 910 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2018. P. 3–15. Communications in Computer and Information Science. DOI: 10.1007/978-3-319-99673-8_1.

13. Popov A., Nikitenko D., Antonov A., Voevodin V. Formal Model of Problems, Methods, Algorithms and Implementations in the Advancing AlgoWiki Open Encyclopedia. CEUR Workshop Proc. Vol. 2281. 2018. P. 1–11. URL: http://ceur-ws.org/Vol-2281/#paper-01.

14. Givens method – Algowiki. URL: https://algowiki-project.org/en/Givens_method (accessed: 23.01.2026).

15. Help:Templates – MediaWiki. URL: https://www.mediawiki.org/wiki/Help:Templates/en (accessed: 23.01.2026).

16. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition. 2001. 672 p.

17. Akl S. The Design and Analysis of Parallel Algorithms. Prentice Hall, 1989. 412 p.

18. Skiena S. The Algorithm Design Manual Second Edition. Springer, 2011. 739 p.

19. Lobanova V., Voronina O., Lobanova N. Theory of Algorithms: A Tutorial. Orel: Orel State University named after I.S. Turgenev, 2017. 95 p. (in Russian).

20. Mehlhorn K., Sanders P. Algorithms and Data Structures: The Basic Toolbox. Springer Science & Business Media, 2008. 305 p. DOI: 10.1007/978-3-540-77978-0.

21. Cormen T., Leiserson C.E., Rivest R.L., Stein C. Introduction To Algorithms, 3rd Edition. Milton, Qld., Jacaranda Wiley, 2009. 1292 p.

22. Sedgewick R. Algorithms in C, Parts 1–4: Fundamentals, Data Structures, Sorting, Searching. Addison-Wesley Professional, 1997. 702 p.

23. Reingold E. Basic Techniques for Design and Analysis of Algorithms. Chapman, Hall/CRC, 2004. 3–1 p. DOI: 10.1201/9780429171529-4.

24. JaJa J. An Introduction to Parallel Algorithms. Addison Wesley Longman Publishing Co., Inc., United States, 1992. 566 p.

25. Alsuwaiyel M. Parallel algorithms. World Scientific Publishers, 2022. 400 p. DOI: 10.1142/12744.

26. Blelloch G., Maggs B. Parallel Algorithms. Computer science handbook / editor-in-chief, Allen B. Tucker–2nd ed. 2004. 10-10–41 p.

27. Selivanova I., Blinov V. Fundamental Algorithms in C++. Construction and Analysis of Data Processing Algorithms: A Tutorial. Ekaterinburg: Ural University Publishing House, 2015. 108 p. (in Russian).

28. McConnell J. Analysis of Algorithms. Jones & Bartlett Learning, 2008. 451 p.

29. Stone doubling algorithm for solving bidiagonal SLAEs. URL: https://algowiki-project.org/en/Stone_doubling_algorithm_for_solving_bidiagonal_SLAEs (accessed: 23.01.2026) (in Russian).

30. Aho A., Hopcroft J., Ullman J. The Design and Analysis of Computer Algorithms. Addison-Wesley Longman Publishing Co., Inc., 1974. 480 p.

31. Antonov A., Maier R., Nikitenko D., Voevodin V. An Approach to Solving the Problem of Supercomputer Co-design. Lobachevskii J Math. 2024. Vol. 45, no. 7. P. 2965–2973. DOI: 10.1134/S1995080224603680.

32. Deterministic algorithm. URL: https://xlinux.nist.gov/dads/HTML/deterministicAlgorithm.html (accessed: 23.01.2026).

33. Smith J. The Design and Analysis of Parallel Algorithms. Oxford University Press, 1993. 470 p.

34. Casanova H., Legrand A., Robert Y. Parallel Algorithms. CRC PRESS, 2020. 335 p.

35. Patterson D., Hennessy J. Computer Organization and Design. 4th Edition. Elsevier, 2012. 919 p.

36. Voevodin V., Antonov A., Nikitenko D., *et al.* Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers and Innovations. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.