Научная статья

УДК 004.021, 004.032.24, 004.051, 004.272
DOI 10.25205/1818-7900-2025-23-2-5-17

Эффективная реализация алгоритмов сортировки с помощью концепции *Q*-детерминанта

Валентина Николаевна Алеева
Егор Константинович Кузнецов

Южно-Уральский государственный университет
Челябинск, Россия

allevavn@susu.ru, https//orcid.org/0000-0002-4717-0045
i-irbis-i@mail.ru, https//orcid.org/0009-0009-1340-0269

Аннотация

Статья содержит исследование возможности эффективной реализации алгоритмов сортировки с помощью концепции *Q*-детерминанта. Для проведения исследования используются алгоритмы шейкерной сортировки, сортировки Шелла, быстрой сортировки и четно-нечетной сортировки слиянием Бэтчера. Для этих алгоритмов получены представления в форме *Q*-детерминантов для сортировки массива, состоящего из небольшого количества элементов. Проведен анализ структуры полученных представлений. На основе результатов анализа описано представление алгоритма сортировки в форме *Q*-детерминанта для общего случая. Рассмотрено применение для алгоритмов сортировки метода проектирования эффективных программ, использующего концепцию *Q*-детерминанта. Применение метода апробировано с помощью разработки эффективных программ, реализующих алгоритм сортировки Шелла на общей и распределенной памяти параллельных вычислительных систем.

Ключевые слова

повышение эффективности параллельных вычислений, *Q*-детерминант алгоритма, представление алгоритма в форме *Q*-детерминанта, *Q*-эффективная реализация алгоритма, ресурс параллелизма алгоритма, *Q*-эффективная программа

Для цитирования

Алеева В. Н., Кузнецов Е. К. Эффективная реализация алгоритмов сортировки с помощью концепции *Q*-детерминанта // Вестник НГУ. Серия: Информационные технологии. 2025. Т. 23, № 2. С. 5–17. DOI 10.25205/1818-7900-2025-23-2-5-17

Effective Implementation of Sorting Algorithms Using the Concept of *Q*-determinant

Valentina N. Aleeva, Egor K. Kuznetsov

South Ural State University,
Chelyabinsk, Russian Federation

allevavn@susu.ru, https//orcid.org/0000-0002-4717-0045
i-irbis-i@mail.ru, https//orcid.org/0009-0009-1340-0269

Abstract

The present paper is devoted to the possibility of efficient implementation for sorting algorithms using the *Q*-determinant concept. We have investigated four algorithms: shaker sort, Shell sort, quicksort and Batcher's odd-even mergesort.

To sort small arrays, we obtained representations in the form of *Q*-determinants for these algorithms. Then the structures of the obtained representations were analyzed and, as a result, for the general case we have described the representations of the sorting algorithms in the form of a *Q*-determinant. Also, for sorting algorithms there was considered the application of the method of designing effective programs using the concept of *Q*-determinant. This application has been tested on shared and distributed memory of parallel computing systems by developing effective programs for the Shell sort.
Keywords
improving parallel computing efficiency, *Q*-determinant of algorithm, representation of algorithm in form of *Q*-determinant, *Q*-effective implementation of algorithm, parallelism resource of algorithm, *Q*-effective program
For citation
Aleeva V. N., Kuznetsov E. K. Effective implementation of sorting algorithms using the concept of *Q*-determinant. *Vestnik NSU. Series: Information Technologies*, 2025, vol. 23, no. 2, pp. 5–17 (in Russ.) DOI 10.25205/1818-7900-2025-23-2-5-17

Введение

Одной из самых распространенных задач в программировании является сортировка. Для ее решения используются различные алгоритмы сортировки. Проблема эффективной реализации алгоритмов сортировки, как и алгоритмов в целом, является актуальной. Эффективная реализация алгоритмов предполагает их распараллеливание. Концепция *Q*-детерминанта – один из подходов к распараллеливанию численных алгоритмов. Она является теоретической основой данного исследования.

Результаты исследований, полученные в настоящее время на основе концепции *Q*-детерминанта, представляют собой один из вариантов решения проблемы эффективной реализации численных алгоритмов, численных методов и алгоритмических проблем на параллельных вычислительных системах (ПВС).

Для создания эффективных программ с помощью концепции *Q*-детерминанта был специально разработан метод проектирования. Создаваемые эффективные программы называются *Q*-эффективными, а сам метод – методом проектирования *Q*-эффективных программ. В настоящее время существует коллекция алгоритмов, к которым был применен этот метод. Особенность данной статьи состоит в том, что она является первой статьей, посвященной исследованию возможности эффективной реализации алгоритмов сортировки с помощью концепции *Q*-детерминанта.

Цель данного исследования заключается в том, чтобы показать возможность эффективной реализации алгоритмов сортировки с помощью концепции *Q*-детерминанта. Для достижения цели должны быть решены следующие задачи.

1. Выявление структуры представлений в форме *Q*-детерминантов алгоритмов сортировки.
2. Описание применения метода проектирования *Q*-эффективных программ для алгоритмов сортировки.
3. Апробация метода проектирования *Q*-эффективных программ на конкретном алгоритме сортировки.

Теоретические основы исследования

Концепция *Q*-детерминанта впервые была изложена в работе [1], а ее развитие и применение описаны в работах [2–9]. Здесь дадим краткое пояснение понятий, используемых в концепции *Q*-детерминанта, основным из которых является понятие *Q*-детерминанта алгоритма.

Определение 1. *Выражение* над множеством входных данных *B* алгоритма и множеством операций *Q*, используемых алгоритмом, определим как терм в стандартном смысле математической логики [10].

Определение 2. *Цепочкой длины n* будем называть выражение, представляющее собой результат применения некоторой ассоциативной операции из *Q* к *n* выражениям.

Пусть $N = n_1, \cdots, n_k$ – множество параметров размерности алгоритмической проблемы, решаемой алгоритмом. Тогда через $\overline{N} = \overline{n}_1, \cdots, \overline{n}_k$ обозначим кортеж, где $\overline{n}_i$ – некоторое заданное значение параметра n_i для каждого $i \in \{1, \cdots, k\}$, а через $\{\overline{N}\}$ – множество всех кортежей $\overline{N}$.

Определим понятия *Q-термов*. *Q*-термы могут быть безусловными, условными и условными бесконечными.

Определение 3. Если $N = \varnothing$, то любое выражение w над B и Q называется *безусловным Q-термом*. Пусть $N \neq \varnothing$ и V – множество всех выражений над B и Q. Любое отображение $w: \overline{N} \to V \cup \varnothing$ также называется *безусловным Q-термом*.

Условные Q-термы состоят из конечного множества пар, а *условные бесконечные Q-термы* – из бесконечного множества пар безусловных *Q*-термов. Первые безусловные *Q*-термы пар принимают значения логического типа, поэтому называются *логическими Q-термами*.

Определение 4. Если алгоритм состоит в том, что для вычисления значения каждой выходной переменной $y_i \left(i \in \{1,..., m\}\right)$ нужно вычислить значение соответствующего *Q*-терма $f_i \left(i \in \{1,..., m\}\right)$, где m – количество выходных переменных, то множество *Q*-термов $\{f_i\}_{i \in \{1,..., m\}}$ называется *Q-детерминантом алгоритма*.

Определение 5. Система уравнений $y_i = f_i \left(i \in \{1,..., m\}\right)$ называется *представлением алгоритма в форме Q-детерминанта*.

Определение 6. Процесс вычисления *Q*-термов $\{f_i\}_{i \in \{1,..., m\}}$ называется *реализацией алгоритма*.

Определение 7. Реализация алгоритма называется *параллельной*, если существуют операции, которые выполняются одновременно.

Определение 8. Реализация алгоритма называется *Q-эффективной*, если *Q*-термы $\{f_i\}_{i \in \{1,..., m\}}$ вычисляются одновременно, операции при их вычислении выполняются по мере готовности, при этом, если несколько операций цепочки готовы к выполнению, то они выполняются по схеме сдваивания.

Замечание. Определение *Q*-эффективной реализации показывает, что она полностью использует ресурс параллелизма алгоритма.

Ресурс параллелизма алгоритма α определяют *высота* D_α и *ширина* P_α *алгоритма*. D_α характеризует время выполнения *Q*-эффективной реализации алгоритма, а P_α – количество вычислителей (вычислительных ядер, процессоров) ПВС, необходимое для выполнения *Q*-эффективной реализации. Кроме того, P_α характеризует масштабируемость алгоритма α. Подробно понятия высоты и ширины алгоритма рассматриваются в работах [2; 3; 7–9].

Определение 9. Реализация алгоритма называется *выполнимой*, если одновременно должно выполняться конечное (непустое) множество операций.

Метод проектирования *Q*-эффективных программ состоит из трех этапов:

1) построение *Q*-детерминанта алгоритма;
2) описание *Q*-эффективной реализации алгоритма;
3) разработка программы для выполнимой *Q*-эффективной реализации алгоритма.

Определение 10. Программу будем называть *Q-эффективной*, если она разработана с помощью данного метода.

Дадим еще одно определение *Q*-эффективной программы.

Определение 11. Программу будем называть *Q-эффективной*, если она выполняет *Q*-эффективную реализацию алгоритма.

Определениям 10 и 11 соответствует одно и то же множество программ. Таким образом, понятие *Q*-эффективной программы можно определять как с помощью определения 10, так и с помощью определения 11. Подробно метод проектирования *Q*-эффективных программ изложен в работах [2; 5].

Представления алгоритмов сортировки в форме *Q*-детерминантов

Для исследования структуры представлений алгоритмов сортировки в форме *Q*-детерминантов использовались алгоритмы шейкерной сортировки, сортировки Шелла, быстрой сортировки и четно-нечетной сортировки слиянием Бэтчера.

Шейкерная сортировка [11, с. 130] является разновидностью пузырьковой сортировки. Она заключается в следующем. Элементы сортируемого массива разбиваются на левую, рабочую, где происходит движение, и правую части. Границы рабочей части массива устанавливаются в месте последнего обмена на каждой итерации. Обработка рабочей части состоит в прямом и обратном проходе. Прямой проход осуществляется слева направо и перемещает максимальный элемент из рабочей части к началу правой части. Обратный проход осуществляется справа налево и перемещает минимальный элемент из рабочей части к концу левой части.

Сортировка Шелла [11, с. 102] является усовершенствованным вариантом сортировки вставками. Идея метода заключается в сравнении разделенных на группы элементов сортируемого массива, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно d, например, $L/2$, где L – общее число элементов массива. На первом шаге каждая группа содержит два элемента, расположенных друг от друга на расстоянии $L/2$, они сравниваются между собой и, если необходимо, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается в два раза, и количество групп, соответственно, уменьшается. На последнем шаге значение d равно единице, и проход по массиву происходит в последний раз.

Алгоритмы быстрой сортировки [11, с. 134] и четно-нечетной сортировки слиянием Бэтчера [11, с. 249] часто используются на практике.

1) $A(1)A(2)A(3)A(4) * (A(1)\le A(2))\&(A(2)\le A(3))\&(A(3)\le A(4))\&(A(2)\le A(3))\&(A(1)\le A(2))\&(A(2)\le A(3))$
2) $A(1)A(2)A(4)A(3) * (A(1)\le A(2))\&(A(2)\le A(3))\&(A(3)>A(4))\&(A(2)\le A(4))\&(A(1)\le A(2))\&(A(2)\le A(4))$
3) $A(1)A(4)A(2)A(3) * (A(1)\le A(2))\&(A(2)\le A(3))\&(A(3)>A(4))\&(A(2)>A(4))\&(A(1)\le A(4))\&(A(4)\le A(2))$
4) $A(4)A(1)A(2)A(3) * (A(1)\le A(2))\&(A(2)\le A(3))\&(A(3)>A(4))\&(A(2)>A(4))\&(A(1)>A(4))\&(A(1)\le A(2))$
5) $A(1)A(3)A(2)A(4) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)\le A(4))\&(A(3)\le A(2))\&(A(1)\le A(3))\&(A(3)\le A(2))$
6) $A(3)A(1)A(2)A(4) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)\le A(4))\&(A(3)\le A(2))\&(A(1)>A(3))\&(A(1)\le A(2))$
7) $A(1)A(3)A(4)A(2) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)>A(4))\&(A(3)\le A(4))\&(A(1)\le A(3))\&(A(3)\le A(4))$
8) $A(3)A(1)A(4)A(2) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)>A(4))\&(A(3)\le A(4))\&(A(1)>A(3))\&(A(1)\le A(4))$
9) $A(3)A(4)A(1)A(2) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)>A(4))\&(A(3)\le A(4))\&(A(1)>A(3))\&(A(1)>A(4))$
10) $A(1)A(4)A(3)A(2) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)>A(4))\&(A(3)>A(4))\&(A(1)\le A(4))\&(A(4)\le A(3))$
11) $A(4)A(1)A(3)A(2) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)>A(4))\&(A(3)>A(4))\&(A(1)>A(4))\&(A(1)\le A(3))$
12) $A(4)A(3)A(1)A(2) * (A(1)\le A(2))\&(A(2)>A(3))\&(A(2)>A(4))\&(A(3)>A(4))\&(A(1)>A(4))\&(A(1)>A(3))$
13) $A(2)A(1)A(3)A(4) * (A(1)>A(2))\&(A(1)\le A(3))\&(A(3)\le A(4))\&(A(1)\le A(3))\&(A(2)\le A(1))\&(A(1)\le A(3))$
14) $A(2)A(1)A(4)A(3) * (A(1)>A(2))\&(A(1)\le A(3))\&(A(3)>A(4))\&(A(1)\le A(4))\&(A(2)\le A(1))\&(A(1)\le A(4))$
15) $A(2)A(4)A(1)A(3) * (A(1)>A(2))\&(A(1)\le A(3))\&(A(3)>A(4))\&(A(1)>A(4))\&(A(2)\le A(4))\&(A(4)\le A(1))$
16) $A(4)A(2)A(1)A(3) * (A(1)>A(2))\&(A(1)\le A(3))\&(A(3)>A(4))\&(A(1)>A(4))\&(A(2)>A(4))\&(A(2)\le A(1))$
17) $A(2)A(3)A(1)A(4) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)\le A(4))\&(A(3)\le A(1))\&(A(2)\le A(3))\&(A(3)\le A(1))$
18) $A(3)A(2)A(1)A(4) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)\le A(4))\&(A(3)\le A(1))\&(A(2)>A(3))\&(A(2)\le A(1))$
19) $A(2)A(3)A(4)A(1) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)>A(4))\&(A(3)\le A(4))\&(A(2)\le A(3))\&(A(3)\le A(4))$
20) $A(3)A(2)A(4)A(1) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)>A(4))\&(A(3)\le A(4))\&(A(2)>A(3))\&(A(2)\le A(4))$
21) $A(3)A(4)A(2)A(1) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)>A(4))\&(A(3)\le A(4))\&(A(2)>A(3))\&(A(2)>A(4))$
22) $A(2)A(4)A(3)A(1) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)>A(4))\&(A(3)>A(4))\&(A(2)\le A(4))\&(A(4)\le A(3))$
23) $A(4)A(2)A(3)A(1) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)>A(4))\&(A(3)>A(4))\&(A(2)>A(4))\&(A(2)\le A(3))$
24) $A(4)A(3)A(2)A(1) * (A(1)>A(2))\&(A(1)>A(3))\&(A(1)>A(4))\&(A(3)>A(4))\&(A(2)>A(4))\&(A(2)>A(3))$

Рис. 1. Представление в форме *Q*-детерминанта алгоритма шейкерной сортировки для массива из четырех элементов
Fig. 1. Representation in the form of a *Q*-determinant of the shaker sort algorithm for a four-element array

Сначала было получено представление в форме *Q*-детерминанта алгоритма шейкерной сортировки для массива из четырех элементов, представленное на рис. 1. Для этого использовалась подробная блок-схема алгоритма шейкерной сортировки в формате JSON, изображенная на рис. 2. Описание блок-схемы алгоритма в формате JSON и алгоритм формирования представления алгоритма в форме *Q*-детерминанта по его блок-схеме приведены в работах [2; 8].


```
{"Vertices":[
        {"Id":1,"Type":0,"Content":"Start"},
        {"Id":2,"Type":4,"Content":"A(1)"},
        {"Id":3,"Type":4,"Content":"A(2)"},
        {"Id":4,"Type":4,"Content":"A(3)"},
        {"Id":5,"Type":4,"Content":"A(4)"},
        {"Id":6,"Type":2,"Content":"I(1)=1"},
        {"Id":7,"Type":2,"Content":"I(2)=2"},
        {"Id":8,"Type":2,"Content":"I(3)=3"},
        {"Id":9,"Type":2,"Content":"I(4)=4"},
        {"Id":10,"Type":3,"Content":"A(I(1))<=A(I(2))"},
        {"Id":11,"Type":2,"Content":"J(1)=I(1)"},
        {"Id":12,"Type":2,"Content":"I(1)=I(2)"},
        {"Id":13,"Type":2,"Content":"I(2)=J(1)"},
        {"Id":14,"Type":3,"Content":"A(I(2))<=A(I(3))"},
        {"Id":15,"Type":2,"Content":"J(2)=I(2)"},
        {"Id":16,"Type":2,"Content":"I(2)=I(3)"},
        {"Id":17,"Type":2,"Content":"I(3)=J(2)"},
        {"Id":18,"Type":3,"Content":"A(I(3))<=A(I(4))"},
        {"Id":19,"Type":2,"Content":"J(3)=I(3)"},
        {"Id":20,"Type":2,"Content":"I(3)=I(4)"},
        {"Id":21,"Type":2,"Content":"I(4)=J(3)"},
        {"Id":22,"Type":3,"Content":"A(I(2))<=A(I(3))"},
        {"Id":23,"Type":2,"Content":"J(2)=I(2)"},
        {"Id":24,"Type":2,"Content":"I(2)=I(3)"},
        {"Id":25,"Type":2,"Content":"I(3)=J(2)"},
        {"Id":26,"Type":3,"Content":"A(I(1))<=A(I(2))"},
        {"Id":27,"Type":2,"Content":"J(1)=I(1)"},
        {"Id":28,"Type":2,"Content":"I(1)=I(2)"},
        {"Id":29,"Type":2,"Content":"I(2)=J(1)"},
        {"Id":30,"Type":3,"Content":"A(I(2))<=A(I(3))"},
        {"Id":31,"Type":2,"Content":"J(2)=I(2)"},
        {"Id":32,"Type":2,"Content":"I(2)=I(3)"},
        {"Id":33,"Type":2,"Content":"I(3)=J(2)"},
        {"Id":34,"Type":4,"Content":"A(I(1))"},
        {"Id":35,"Type":4,"Content":"A(I(2))"},
        {"Id":36,"Type":4,"Content":"A(I(3))"},
        {"Id":37,"Type":4,"Content":"A(I(4))"},
        {"Id":38,"Type":1,"Content":"End"}],
"Edges":[
        {"From":1,"To":2,"Type":2},     {"From":2,"To":3,"Type":2},
        {"From":3,"To":4,"Type":2},     {"From":4,"To":5,"Type":2},
        {"From":5,"To":6,"Type":2},     {"From":6,"To":7,"Type":2},
        {"From":7,"To":8,"Type":2},     {"From":8,"To":9,"Type":2},
        {"From":9,"To":10,"Type":2},    {"From":10,"To":14,"Type":1},
        {"From":10,"To":11,"Type":0},   {"From":11,"To":12,"Type":2},
        {"From":12,"To":13,"Type":2},   {"From":13,"To":14,"Type":2},
        {"From":14,"To":18,"Type":1},   {"From":14,"To":15,"Type":0},
        {"From":15,"To":16,"Type":2},   {"From":16,"To":17,"Type":2},
        {"From":17,"To":18,"Type":2},   {"From":18,"To":22,"Type":1},
        {"From":18,"To":19,"Type":0},   {"From":19,"To":20,"Type":2},
        {"From":20,"To":21,"Type":2},   {"From":21,"To":22,"Type":2},
        {"From":22,"To":26,"Type":1},   {"From":22,"To":23,"Type":0},
        {"From":23,"To":24,"Type":2},   {"From":24,"To":25,"Type":2},
        {"From":25,"To":26,"Type":2},   {"From":26,"To":30,"Type":1},
        {"From":26,"To":27,"Type":0},   {"From":27,"To":28,"Type":2},
        {"From":28,"To":29,"Type":2},   {"From":29"To":30,"Type":2},
        {"From":30,"To":34,"Type":1},   {"From":30,"To":31,"Type":0},
        {"From":31,"To":32,"Type":2},   {"From":32"To":33,"Type":2},
        {"From":33,"To":34,"Type":2},   {"From":34,"To":35,"Type":2},
        {"From":35"To":36,"Type":2},    {"From":36,"To":37,"Type":2},
        {"From":37"To":38,"Type":2}]}
```


Рис. 2. Блок-схема алгоритма шейкерной сортировки в формате JSON для массива из четырех элементов
Fig. 2. Shaker sort algorithm flowchart in JSON format for a four-element array

Аналогично для массива из четырех элементов с помощью блок-схем в формате JSON были получены представления в форме Q-детерминантов алгоритмов сортировки Шелла, быстрой сортировки и четно-нечетной сортировки слиянием Бэтчера. Они показаны на рис. 3, 4 и 5. На всех пяти рисунках через $A(1)$, $A(2)$, $A(3)$, $A(4)$ обозначены элементы сортируемого массива.

1) A(1)A(2)A(3)A(4) * (A(1)≤A(3))&(A(2)≤A(4))&(A(1)≤A(2))&(A(2)≤A(3))&(A(3)≤A(4))
2) A(1)A(3)A(2)A(4) * (A(1)≤A(3))&(A(2)≤A(4))&(A(1)≤A(2))&(A(2)>A(3))&(A(1)≤A(3))&(A(2)≤A(4))
3) A(2)A(1)A(3)A(4) * (A(1)≤A(3))&(A(2)≤A(4))&(A(1)>A(2))&(A(1)≤A(3))&(A(3)≤A(4))
4) A(1)A(4)A(3)A(2) * (A(1)≤A(3))&(A(2)>A(4))&(A(1)≤A(4))&(A(4)≤A(3))&(A(3)≤A(2))
5) A(1)A(3)A(4)A(2) * (A(1)≤A(3))&(A(2)>A(4))&(A(1)≤A(4))&(A(4)>A(3))&(A(1)≤A(3))&(A(4)≤A(2))
6) A(4)A(1)A(3)A(2) * (A(1)≤A(3))&(A(2)>A(4))&(A(1)>A(4))&(A(1)≤A(3))&(A(3)≤A(2))
7) A(3)A(2)A(1)A(4) * (A(1)>A(3))&(A(2)≤A(4))&(A(3)≤A(2))&(A(2)≤A(1))&(A(1)≤A(4))
8) A(3)A(1)A(2)A(4) * (A(1)>A(3))&(A(2)≤A(4))&(A(3)≤A(2))&(A(2)>A(1))&(A(3)≤A(1))&(A(2)≤A(4))
9) A(2)A(3)A(1)A(4) * (A(1)>A(3))&(A(2)≤A(4))&(A(3)>A(2))&(A(3)≤A(1))&(A(1)≤A(4))
10) A(3)A(4)A(1)A(2) * (A(1)>A(3))&(A(2)>A(4))&(A(3)≤A(4))&(A(4)≤A(1))&(A(1)≤A(2))
11) A(3)A(1)A(4)A(2) * (A(1)>A(3))&(A(2)>A(4))&(A(3)≤A(4))&(A(4)>A(1))&(A(3)≤A(1))&(A(4)≤A(2))
12) A(4)A(3)A(1)A(2) * (A(1)>A(3))&(A(2)>A(4))&(A(3)>A(4))&(A(3)≤A(1))&(A(1)≤A(2))
13) A(1)A(2)A(4)A(3) * (A(1)≤A(3))&(A(2)≤A(4))&(A(1)≤A(2))&(A(2)≤A(3))&(A(3)>A(4))&(A(2)≤A(4))
14) A(2)A(1)A(4)A(3) * (A(1)≤A(3))&(A(2)≤A(4))&(A(1)>A(2))&(A(1)≤A(3))&(A(3)>A(4))&(A(1)≤A(4))
15) A(1)A(4)A(2)A(3) * (A(1)≤A(3))&(A(2)>A(4))&(A(1)≤A(4))&(A(4)≤A(3))&(A(3)>A(2))&(A(4)≤A(2))
16) A(4)A(1)A(2)A(3) * (A(1)≤A(3))&(A(2)>A(4))&(A(1)>A(4))&(A(1)≤A(3))&(A(3)>A(2))&(A(1)≤A(2))
17) A(3)A(2)A(4)A(1) * (A(1)>A(3))&(A(2)≤A(4))&(A(3)≤A(2))&(A(2)≤A(1))&(A(1)>A(4))&(A(2)≤A(4))
18) A(2)A(3)A(4)A(1) * (A(1)>A(3))&(A(2)≤A(4))&(A(3)>A(2))&(A(3)≤A(1))&(A(1)>A(4))&(A(3)≤A(4))
19) A(3)A(4)A(2)A(1) * (A(1)>A(3))&(A(2)>A(4))&(A(3)≤A(4))&(A(4)≤A(1))&(A(1)>A(2))&(A(4)≤A(2))
20) A(4)A(3)A(2)A(1) * (A(1)>A(3))&(A(2)>A(4))&(A(3)>A(4))&(A(3)≤A(1))&(A(1)>A(2))&(A(3)≤A(2))
21) A(2)A(4)A(1)A(3) * (A(1)≤A(3))&(A(2)≤A(4))&(A(1)>A(2))&(A(1)≤A(3))&(A(3)>A(4))&(A(1)>A(4))&(A(2)≤A(4))
22) A(4)A(2)A(1)A(3) * (A(1)≤A(3))&(A(2)>A(4))&(A(1)>A(4))&(A(1)≤A(3))&(A(3)>A(2))&(A(1)>A(2))&(A(4)≤A(2))
23) A(2)A(4)A(3)A(1) * (A(1)>A(3))&(A(2)≤A(4))&(A(3)>A(2))&(A(3)≤A(1))&(A(1)>A(4))&(A(3)>A(4))&(A(2)≤A(4))
24) A(4)A(2)A(3)A(1) * (A(1)>A(3))&(A(2)>A(4))&(A(3)>A(4))&(A(3)≤A(1))&(A(1)>A(2))&(A(3)>A(2))&(A(4)≤A(2))

Рис. 3. Представление в форме Q-детерминанта алгоритма сортировки Шелла для массива из четырех элементов
Fig. 3. Representation in the form of a Q-determinant of the Shell sort algorithm for a four-element array

1) A(1)A(4)A(2)A(3) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(2)≤A(3))&(A(2)>A(4))
2) A(1)A(3)A(2)A(4) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(2)>A(3))&(A(2)≤A(4))
3) A(1)A(2)A(3)A(4) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(2)≤A(3))&(A(2)≤A(4))&(A(3)≤A(4))
4) A(1)A(2)A(4)A(3) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(2)≤A(3))&(A(2)≤A(4))&(A(3)>A(4))
5) A(1)A(3)A(4)A(2) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(2)>A(3))&(A(2)>A(4))&(A(3)≤A(4))
6) A(1)A(4)A(3)A(2) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(2)>A(3))&(A(2)>A(4))&(A(3)>A(4))
7) A(4)A(1)A(2)A(3) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)>A(4))&(A(2)≤A(3))
8) A(4)A(1)A(3)A(2) * (A(1)≤A(2))&(A(1)≤A(3))&(A(1)>A(4))&(A(2)>A(3))
9) A(3)A(1)A(2)A(4) * (A(1)≤A(2))&(A(1)>A(3))&(A(1)≤A(4))&(A(2)≤A(4))
10) A(3)A(1)A(4)A(2) * (A(1)≤A(2))&(A(1)>A(3))&(A(1)≤A(4))&(A(2)>A(4))
11) A(3)A(4)A(1)A(2) * (A(1)≤A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(3)≤A(4))
12) A(4)A(3)A(1)A(2) * (A(1)≤A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(3)>A(4))
13) A(2)A(1)A(3)A(4) * (A(1)>A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(3)≤A(4))
14) A(2)A(1)A(4)A(3) * (A(1)>A(2))&(A(1)≤A(3))&(A(1)≤A(4))&(A(3)>A(4))
15) A(2)A(4)A(1)A(3) * (A(1)>A(2))&(A(1)≤A(3))&(A(1)>A(4))&(A(2)≤A(4))
16) A(4)A(2)A(1)A(3) * (A(1)>A(2))&(A(1)≤A(3))&(A(1)>A(4))&(A(2)>A(4))
17) A(2)A(3)A(1)A(4) * (A(1)>A(2))&(A(1)>A(3))&(A(1)≤A(4))&(A(2)≤A(3))
18) A(3)A(2)A(1)A(4) * (A(1)>A(2))&(A(1)>A(3))&(A(1)≤A(4))&(A(2)>A(3))
19) A(4)A(2)A(3)A(1) * (A(1)>A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(2)≤A(3))&(A(2)>A(4))
20) A(3)A(2)A(4)A(1) * (A(1)>A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(2)>A(3))&(A(2)≤A(4))
21) A(2)A(3)A(4)A(1) * (A(1)>A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(2)≤A(3))&(A(2)≤A(4))&(A(3)≤A(4))
22) A(2)A(4)A(3)A(1) * (A(1)>A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(2)≤A(3))&(A(2)≤A(4))&(A(3)>A(4))
23) A(3)A(4)A(2)A(1) * (A(1)>A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(2)>A(3))&(A(2)>A(4))&(A(3)≤A(4))
24) A(4)A(3)A(2)A(1) * (A(1)>A(2))&(A(1)>A(3))&(A(1)>A(4))&(A(2)>A(3))&(A(2)>A(4))&(A(3)>A(4))

Рис. 4. Представление в форме Q-детерминанта алгоритма быстрой сортировки для массива из четырех элементов
Fig. 4. Representation in the form of a Q-determinant of quicksort algorithm for a four-element array

Поясним формат представления алгоритмов сортировки в форме Q-детерминантов, используемый на рис. 1, 3, 4 и 5. Q-детерминанты состоят из четырех условных Q-термов длины 4! = 24, равной количеству всех перестановок элементов массива $A(1)$, $A(2)$, $A(3)$, $A(4)$.

1) A(1)A(2)A(3)A(4) * (A(1)≤A(2))&(A(3)≤A(4))&(A(1)≤A(3))&(A(2)≤A(4))&(A(2)≤A(3))
2) A(1)A(3)A(2)A(4) * (A(1)≤A(2))&(A(3)≤A(4))&(A(1)≤A(3))&(A(2)≤A(4))&(A(2)>A(3))
3) A(1)A(3)A(4)A(2) * (A(1)≤A(2))&(A(3)≤A(4))&(A(1)≤A(3))&(A(2)>A(4))&(A(4)>A(3))
4) A(3)A(1)A(2)A(4) * (A(1)≤A(2))&(A(3)≤A(4))&(A(1)>A(3))&(A(2)≤A(4))&(A(2)>A(1))
5) A(3)A(4)A(1)A(2) * (A(1)≤A(2))&(A(3)≤A(4))&(A(1)>A(3))&(A(2)>A(4))&(A(3)≤A(1))
6) A(3)A(1)A(4)A(2) * (A(1)≤A(2))&(A(3)≤A(4))&(A(1)>A(3))&(A(2)>A(4))&(A(4)>A(1))
7) A(1)A(2)A(4)A(3) * (A(1)≤A(2))&(A(3)>A(4))&(A(1)≤A(4))&(A(2)≤A(3))&(A(2)≤A(4))
8) A(1)A(4)A(2)A(3) * (A(1)≤A(2))&(A(3)>A(4))&(A(1)≤A(4))&(A(2)≤A(3))&(A(2)>A(4))
9) A(1)A(4)A(3)A(2) * (A(1)≤A(2))&(A(3)>A(4))&(A(1)≤A(4))&(A(2)>A(3))&(A(3)>A(4))
10) A(4)A(1)A(2)A(3) * (A(1)≤A(2))&(A(3)>A(4))&(A(1)>A(4))&(A(2)≤A(3))&(A(2)>A(1))
11) A(4)A(3)A(1)A(2) * (A(1)≤A(2))&(A(3)>A(4))&(A(1)>A(4))&(A(2)>A(3))&(A(3)≤A(1))
12) A(4)A(1)A(3)A(2) * (A(1)≤A(2))&(A(3)>A(4))&(A(1)>A(4))&(A(2)>A(3))&(A(3)>A(1))
13) A(2)A(1)A(3)A(4) * (A(1)>A(2))&(A(3)≤A(4))&(A(2)≤A(3))&(A(1)≤A(4))&(A(1)≤A(3))
14) A(2)A(3)A(1)A(4) * (A(1)>A(2))&(A(3)≤A(4))&(A(2)≤A(3))&(A(1)≤A(4))&(A(1)>A(3))
15) A(2)A(3)A(4)A(1) * (A(1)>A(2))&(A(3)≤A(4))&(A(2)≤A(3))&(A(1)>A(4))&(A(4)>A(3))
16) A(3)A(2)A(1)A(4) * (A(1)>A(2))&(A(3)≤A(4))&(A(2)>A(3))&(A(1)≤A(4))&(A(1)>A(2))
17) A(3)A(4)A(2)A(1) * (A(1)>A(2))&(A(3)≤A(4))&(A(2)>A(3))&(A(1)>A(4))&(A(4)≤A(2))
18) A(3)A(2)A(4)A(1) * (A(1)>A(2))&(A(3)≤A(4))&(A(2)>A(3))&(A(1)>A(4))&(A(4)>A(2))
19) A(2)A(1)A(4)A(3) * (A(1)>A(2))&(A(3)>A(4))&(A(2)≤A(4))&(A(1)≤A(3))&(A(1)≤A(4))
20) A(2)A(4)A(1)A(3) * (A(1)>A(2))&(A(3)>A(4))&(A(2)≤A(4))&(A(1)≤A(3))&(A(1)>A(4))
21) A(2)A(4)A(3)A(1) * (A(1)>A(2))&(A(3)>A(4))&(A(2)≤A(4))&(A(1)>A(3))&(A(3)>A(4))
22) A(4)A(2)A(1)A(3) * (A(1)>A(2))&(A(3)>A(4))&(A(2)>A(4))&(A(1)≤A(3))&(A(1)>A(2))
23) A(4)A(3)A(2)A(1) * (A(1)>A(2))&(A(3)>A(4))&(A(2)>A(4))&(A(1)>A(3))&(A(3)≤A(2))
24) A(4)A(2)A(3)A(1) * (A(1)>A(2))&(A(3)>A(4))&(A(2)>A(4))&(A(1)>A(3))&(A(3)>A(2))

Рис. 5. Представление в форме Q-детерминанта алгоритма четно-нечетной сортировки слиянием Бэтчера для массива из четырех элементов

Fig. 5. Representation in the form of a Q-determinant of Batcher's odd-even mergesort algorithm for a four-element array

Рисунки содержат представления в форме Q-детерминантов всех четырех алгоритмов не в форме, используемой в определении 5, а в компактной форме. Каждая строка компактной формы включает:

1) порядковый номер строки, который соответствует значению номера пары $j\left(j \in \{1,\ldots,4!\}\right)$ условного Q-терма, то есть одной перестановке элементов сортируемого массива;

2) саму перестановку $w_j^i\left(i \in \{1,2,3,4\}\right)$ элементов сортируемого массива;

3) логический Q-терм u_j, отделенный от перестановки знаком «*».

Компактная запись представления в форме Q-детерминанта удобна для проведения анализа структуры логических Q-термов. Также для удобства проведения анализа специально использована небольшая длина сортируемого массива.

На рис. 1, 3–5 хорошо видно, что структура логических Q-термов зависит от алгоритма сортировки. Каждый логический Q-терм представляет собой цепочку, являющуюся результатом применения операции конъюнкции к различным попарным сравнениям сортируемых элементов.

На основе анализа сформированных по блок-схемам представлений в форме *Q*-детерминантов для четырех алгоритмов сортировки получаем следующее описание представления алгоритма сортировки в форме *Q*-детерминанта для общего случая. Обозначим через L число элементов сортируемого массива, через $A(i)\left(i \in \{1,\ldots,L\}\right)$ – начальное расположение элементов сортируемого массива, а через $C(i)\left(i \in \{1,\ldots,L\}\right)$ – элементы отсортированного массива. Тогда, в соответствии с определением 5, представление алгоритма сортировки в форме *Q*-детерминанта имеет вид $C(i) = \left\{\left(u_j, w_j^i\right)\right\}$, где $i \in \{1,\ldots,L\}$, $j \in \{1,\ldots,L!\}, \left\{\left(u_j, w_j^i\right)\right\}$, – условный *Q*-терм длины L!. Для любого $j \in \{1,\ldots,L!\}$ логические *Q*-термы u_j одинаковы во всех условных *Q*-термах, а $w_j^i\left(i \in \{1,\ldots,L\}\right)$ представляют собой одну из возможных перестановок элементов сортируемого массива $A(i)\left(i \in \{1,\ldots,L\}\right)$. Если логический *Q*-терм u_j имеет значение *true*, то отсортированный массив будет являться перестановкой $w_j^i\left(i \in \{1,\ldots,L\}\right)$.

Оценка ресурсов параллелизма алгоритмов сортировки

Оценим ресурсы параллелизма всех четырех проанализированных алгоритмов сортировки для массива из четырех элементов. Высота и ширина алгоритмов являются функциями от параметров размерности алгоритмической проблемы, которую решает алгоритм, если параметры размерности существуют. В нашем случае алгоритмическая проблема, заключающаяся в сортировке массива из чисел, имеет один параметр размерности – длину массива. По полученным представлениям в форме *Q*-детерминантов для параметра размерности 4 легко вычислить высоту и ширину всех четырех алгоритмов.

Для оценки высоты алгоритмов сортировки при любом значении длины массива нужно вычислить количество уровней вложенности каждого из выражений, описывающих логические *Q*-термы, и найти из них максимальное значение. При определении уровней вложенности необходимо учитывать, что цепочки операций в выражениях должны вычисляться по схеме сдваивания. Вычисление показывает, что высота проанализированных алгоритмов при длине массива 4 одинакова и равна четырем.

Для оценки ширины алгоритмов сортировки при любом значении длины массива нужно вычислить количество операций на каждом из уровней вложенности всех выражений, описывающих логические *Q*-термы, и найти из них максимальное значение. Максимальное количество операций имеет первый уровень вложенности. Вычисление показывает, что при длине массива 4 ширина алгоритма шейкерной сортировки равна 144, сортировки Шелла – 140, быстрой сортировки – 116, четно-нечетной сортировки слиянием Бэтчера – 120.

Высота рассмотренных алгоритмов при длине массива 4 одинакова, поэтому примерно одинаково время выполнения реализующих их *Q*-эффективных программ с одинаковыми вычислительными инфраструктурами, т. е. условиями разработки и выполнения программ. Ширина алгоритмов разная, поэтому для выполнения *Q*-эффективных реализаций алгоритмов потребуется разное количество вычислителей ПВС. Учитывая ширину алгоритмов, можно сделать вывод, что все они масштабируемые.

Применение метода проектирования *Q*-эффективных программ для алгоритмов сортировки

Опишем, как применить метод проектирования *Q*-эффективных программ для алгоритмов сортировки. На первом этапе метода должен быть построен *Q*-детерминант алгоритма. Это было уже сделано ранее.

На втором этапе метода нужно описать Q-эффективную реализацию алгоритма. В случае алгоритмов сортировки описание заключается в том, что операции должны выполняться в соответствии с алгоритмом сортировки по мере их готовности к выполнению, при этом, если несколько операций цепочки готовы к выполнению, то они должны выполняться по схеме сдваивания. При такой реализации обеспечивается одновременное вычисление всех Q-термов $\{f_i\}_{i\in\{1,\ldots,L\}}$, составляющих Q-детерминант алгоритма сортировки. Более того, обеспечивается одновременное вычисление всех логических Q-термов $u_j\left(j\in\{1,\ldots,L!\}\right)$ и определение элементов соответствующих им перестановок сортируемого массива $w_j^i\left(i\in\{1,\ldots,L\}\right)$. Таким образом, описанная реализация алгоритма сортировки по определению 8 является Q-эффективной. Конкретное описание Q-эффективной реализации алгоритма сортировки зависит от алгоритма.

На третьем этапе метода должна быть разработана Q-эффективная программа, выполняющая Q-эффективную реализацию алгоритма, если Q-эффективная реализация выполнима. По определению 9 Q-эффективная реализация любого алгоритма сортировки выполнима, так как одновременно должно выполняться конечное (непустое) множество операций. При разработке Q-эффективной программы для общей памяти нужно использовать описание Q-эффективной реализации алгоритма, полученное на втором этапе. При разработке Q-эффективной программы для распределенной памяти нужно также выполнить распределение вычислений по узлам ПВС.

Примеры применения метода проектирования Q-эффективных программ к другим алгоритмам описаны в работах [2; 4; 5; 7; 9].

Эффективная реализация алгоритма сортировки Шелла

Метод проектирования Q-эффективных программ был применен для Q-эффективной реализации алгоритма сортировки Шелла на общей и распределенной памяти ПВС. Исследования проводились на суперкомпьютере «Торнадо» Южно-Уральского государственного университета. При разработке программ применялся язык программирования C++. Для общей памяти использовался один вычислительный узел, который содержит два центральных процессора Intel Xeon X5680 с частотой 3,33 GHz, каждый из которых имеет 6 ядер и поддерживает 12 потоков, оперативная память узла 24 Гб ECC DDR3 Full buffered. При разработке программ применялась технология OpenMP для управления потоками программы. Для распределенной памяти использовалось несколько вычислительных узлов. При разработке программ применялись технологии OpenMP и MPI для управления процессами программы. Обе Q-эффективные программы разработаны так, что все операции над массивами при достаточных вычислительных ресурсах должны выполняться по мере их готовности.

Q-эффективная программа для реализации алгоритма на общей памяти в начале выполнения генерирует случайный, обратный (значения элементов массива не возрастают) и частично отсортированный массивы, затем выполняет их сортировку с применением алгоритма Шелла, после чего фиксирует время выполнения алгоритма для каждого из массивов и записывает эти данные в отдельный файл.

Q-эффективная программа для реализации алгоритма на распределенной памяти в начале выполнения инициализирует MPI, после этого определяет ранг текущего процесса и общее число процессов, затем главный процесс генерирует случайный, обратный и частично отсортированный массивы. После этого сгенерированные массивы разделяются на подмассивы равной длины для каждого процесса программы. Затем осуществляется локальная сортировка подмассивов каждым из процессов с применением алгоритма Шелла. После локальной сортировки главный процесс осуществляет сбор и слияние подмассивов в глобальный отсортированный массив, после чего программа фиксирует время выполнения алгоритма для каждого из массивов и записывает эти данные в отдельный файл.

Разработанные Q-эффективные программы полностью используют ресурс параллелизма алгоритмов. Однако ресурсов вычислительной системы было недостаточно для использования всего параллелизма при вычислительных экспериментах. В этих условиях были оценены динамические характеристики программ. Ускорение программы вычислялось по формуле

$$S_p = T_1 / T_p,$$

где T_1 – время выполнения программы на одном вычислительном ядре; T_p – время выполнения программы на вычислительных ядрах. Затем для вычисления эффективности программы использовалась формула

$$E_p = S_p / p,$$

где S_p – ускорение программы; p – количество используемых вычислительных ядер.

Под последовательной программой мы понимаем параллельную программу, выполняемую на одном ядре одного вычислительного узла. В остальных случаях во время вычислительных экспериментов с программой для общей памяти количество процессорных ядер варьировалось от 2 до 12, а с программой для распределенной памяти на каждом вычислительном узле использовались все 12 физических ядер и максимально возможное количество одновременно выполняющихся нитей, равное 24.

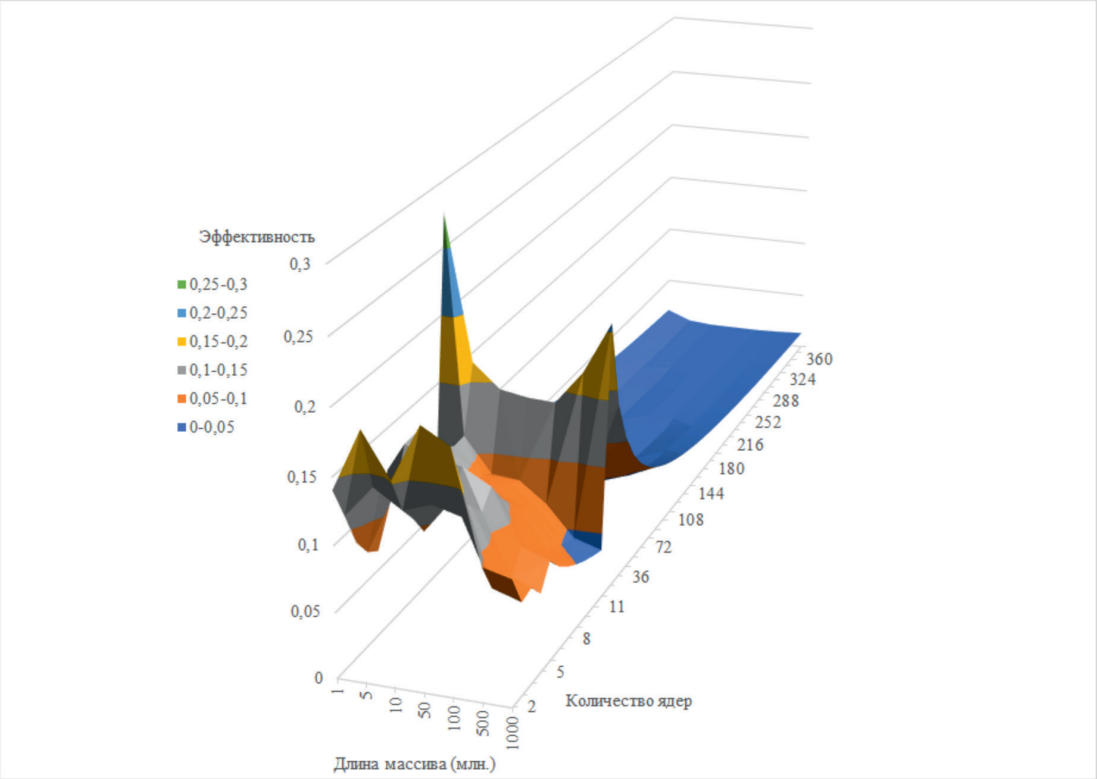

Рис. 6. Эффективность Q-эффективных программ при сортировке случайных массивов
Fig. 6. Efficiency of Q-effective programs in sorting random arrays

На рис. 6–8 представлены характеристики эффективности разработанных Q-эффективных программ при различном упорядочении элементов сортируемого массива. На каждом из рисунков совмещены характеристики Q-эффективных программ для общей и распределенной памяти.

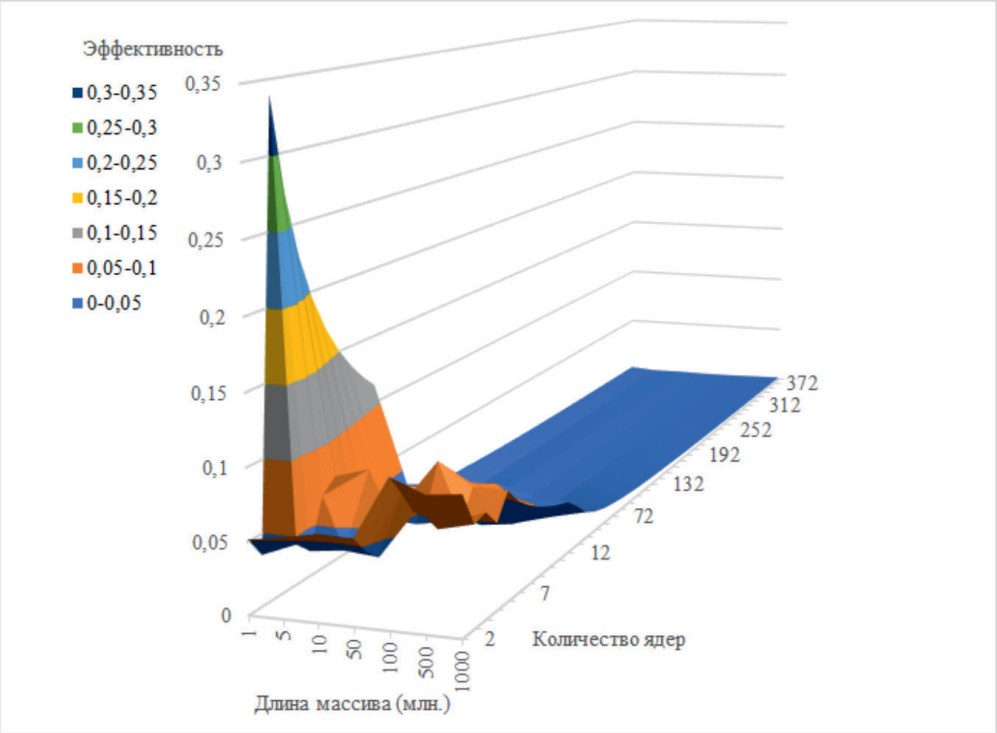

Рис. 7. Эффективность *Q*-эффективных программ при сортировке обратных массивов
Fig. 7. Efficiency of *Q*-effective programs when sorting inverse arrays

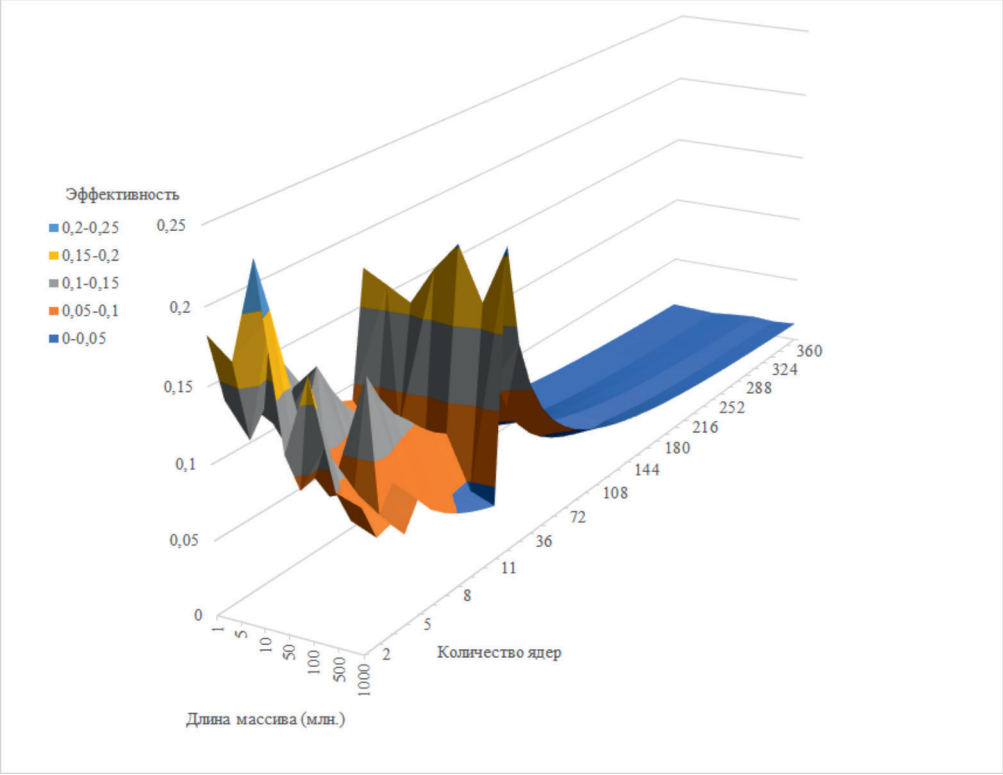

Рис. 8. Эффективность *Q*-эффективных программ при сортировке частично упорядоченных массивов
Fig. 8. Efficiency of *Q*-effective programs in sorting partially ordered arrays

Заключение

Эта статья является первой, в которой концепция *Q*-детерминанта применяется к алгоритмам сортировки. В ней исследована возможность эффективной реализации алгоритмов сортировки с помощью концепции *Q*-детерминанта. При этом выявлена структура представлений в форме *Q*-детерминантов алгоритмов сортировки для общего случая, описано применение метода проектирования *Q*-эффективных программ для алгоритмов сортировки, выполнена апробация метода проектирования *Q*-эффективных программ на конкретном алгоритме сортировки.

Список литературы

1. **Алеева В. Н.** Анализ параллельных численных алгоритмов. Препринт № 590. Новосибирск: ВЦ СО АН СССР, 1985. 23 с.
2. **Aleeva V., Aleev R.** Investigation and Implementation of Parallelism Resources of Numerical Algorithms // ACM Transactions on Parallel Computing. 2023. Vol. 10. No. 2. P. 1–64. DOI: 10.1145/3583755
3. **Алеева В. Н., Зотова П. С., Склезнев Д. С.** Расширение возможностей исследования ресурса параллелизма численных алгоритмов с помощью программной *Q*-системы // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 2. С. 66–81. DOI: 10.14529/cmse210205
4. **Алеева В. Н., Шатов М. Б.** Применение концепции *Q*-детерминанта для эффективной реализации численных алгоритмов на примере метода сопряженных градиентов для решения систем линейных уравнений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 3. С. 56–71. DOI: 10.14529/cmse210304
5. **Aleeva V.** Designing a Parallel Programs on the Base of the Conception of *Q*-Determinant // Supercomputing. RuSCDays 2018. Communications in Computer and Information Science. 2019. Vol. 965. P. 565–577. DOI: 10.1007/978-3-030-05807-4_48
6. **Aleeva V. N., Sharabura I. S., Suleymanov D. E.** Software System for Maximal Parallelization of Algorithms on the Base of the Conception of *Q*-determinant // Parallel Computing Technologies (PaCT 2015). Lecture Notes in Computer Science. 2015. Vol. 9251. P. 3–9. DOI: 10.1007/978-3-319-21909-7_1
7. **Aleeva V. N., Aleev R. Zh.** High-Performance Computing Using Application of *Q*-determinant of Numerical Algorithms // Proceedings – 2018 Global Smart Industry Conference, GloSIC 2018. IEEE. 2018. 8 p. Article number 8570160. DOI: 10.1109/GloSIC.2018.8570160
8. **Aleeva V., Bogatyreva E., Skleznev A. et al.** Software *Q*-system for the Research of the Resource of Numerical Algorithms Parallelism // Supercomputing. RuSCDays 2019. Communications in Computer and Information Science. 2019. Vol. 1129. P. 641–652. DOI: 10.1007/978-3-030-36592-9_52
9. **Aleeva V. N.** Improving Parallel Computing Efficiency // Proceedings – 2020 Global Smart Industry Conference, GloSIC 2020. IEEE. 2020. P. 113–120. Article number 9267828. DOI: 10.1109/GloSIC50886.2020.9267828
10. **Ершов Ю. Л., Палютин Е. А.** Математическая логика. М.: Наука, 1987. 336 с.
11. **Кнут Д.** Искусство программирования. Т. 3. Сортировка и поиск, 2-е изд. М.: Вильямс, 2018. 832 с.

References

1. **Aleeva V. N.** Analysis of Parallel Numerical Algorithms. Preprint no. 590. Novosibirsk, Computing Center of the Siberian Branch of the Academy of Sciences of the USSR, 1985, 23 p. (in Russ.)

2. **Aleeva V., Aleev R.** Investigation and Implementation of Parallelism Resources of Numerical Algorithms. *ACM Transactions on Parallel Computing*, 2023, vol. 10, no. 2, pp. 1–64. DOI: 10.1145/3583755
3. **Aleeva V. N., Zotova P. S., Skleznev D. S.** Advancement of Research for the Parallelism Resource of Numerical Algorithms with the Help of Software *Q-system. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering*, 2021, vol. 10, no. 2, pp. 66–81. (in Russ.) DOI: 10.14529/cmse210205
4. **Aleeva V. N., Shatov M. B.** Application of the Q-determinant Concept for Efficient Implementation of Numerical Algorithms by the Example of the Conjugate Gradient Method for Solving Systems of Linear Equations. Bulletin of the *South Ural State University. Series: Computational Mathematics and Software Engineering*, 2021, vol. 10, no. 3, pp. 56–71. (in Russ.) DOI: 10.14529/cmse210304
5. **Aleeva V.** Designing a Parallel Programs on the Base of the Conception of Q-Determinant. Supercomputing. RuSCDays 2018. *Communications in Computer and Information Science*, 2019, vol. 965, pp. 565–577. DOI: 10.1007/978-3-030-05807-4_48
6. **Aleeva V. N., Sharabura I. S., Suleymanov D. E.** Software System for Maximal Parallelization of Algorithms on the Base of the Conception of Q-determinant. Parallel Computing Technologies (PaCT 2015). *Lecture Notes in Computer Science*, 2015, vol. 9251, pp. 3–9. DOI: 10.1007/978-3-319-21909-7_1
7. **Aleeva V. N., Aleev R. Zh.** High-Performance Computing Using Application of Q-determinant of Numerical Algorithms. In: *Proceedings – 2018 Global Smart Industry Conference, GloSIC 2018. IEEE*, 2018, 8 p. Article number 8570160. DOI: 10.1109/GloSIC.2018.8570160
8. **Aleeva V., Bogatyreva E., Skleznev A. et al.** Software Q-system for the Research of the Resource of Numerical Algorithms Parallelism. Supercomputing. RuSCDays 2019. *Communications in Computer and Information Science*, 2019, vol. 1129, pp. 641–652. DOI: 10.1007/978-3-030-36592-9_52
9. **Aleeva V. N.** Improving Parallel Computing Efficiency. In: *Proceedings – 2020 Global Smart Industry Conference, GloSIC 2020. IEEE*, 2020. P. 113–120. Article number 9267828. DOI: 10.1109/GloSIC50886.2020.9267828
10. **Ershov Yu. L., Palyutin E. A.** Mathematical Logic. Moscow, Mir publ., 1984, 303 p. (in Russ.)
11. **Knuth D.** The Art of Computer Programming. Vol. 3. Sorting and Searching, Second Edition. Moscow, LLC “I.D. Williams”, 2018, 832 p. (in Russ.)

Информация об авторах

Алеева Валентина Николаевна, кандидат физико-математических наук

Кузнецов Егор Константинович, студент магистратуры

Information about the Authors

Valentina N. Aleeva, Candidate of Science in Physics and Mathematics

Egor K. Kuznetsov, Master’s Degree Student

Статья поступила в редакцию 04.03.2025;
одобрена после рецензирования 03.04.2025; принята к публикации 03.04.2025

The article was submitted 04.03.2025;
approved after reviewing 03.04.2025; accepted for publication 03.04.2025