Кубическая Сортировка[1]: оптимальный алгоритм для практически реализуемых Параллельных Компьютеров

Cubesort An Optimal Sorting Algorithm for Feasible Parallel Computers

R. Cypher
University of Washington, Seattle, WA 98195
J.L.C. Sanz
IBM Almaden Research Center, San Jose, CA 95120

Перевод с англ. С. Синцова

Аннотация

В данной работе исследуется задача сортировки N элементов на P-процессорной параллельной машине, где $N \geq P$. Центральным результатом работы, является новый алгоритм, который называется «кубической сортировкой» (КС)[2] и сортирует $N = P^{1+1/k}$ элементов за время $O\left(k \cdot P^{1/k} \cdot \log(P)\right)$, используя P-процессорные компьютеры с замещающе-тасовочной[3] топологией (КОТА)[4]. Таким образом, при любой положительной константе k, КС дает асимптотически оптимальное увеличение скорости по сравнению с последовательной сортировкой. Также, КС упорядочивает $N = P \cdot \log P$ элементов, на P-процессорных КОТА за время $O(\log^3(P)/\log(\log(P)))$. Оба представленных результата быстрее всех ранее опубликованных алгоритмов решения данной задачи. Также, КС дает асимптотически оптимальные алгоритмы сортировки для широкого круга параллельных компьютеров, влкючая кубическую сеть с циклами[5] и гиперкуб. Важным расширением центрального результата работы является алгоритм, который имитирует один шаг Priority-CRCW PRAM[6] с N процессорами и N словами памяти P-процессорном КОТА за время $O\left(k \cdot P^{1/k} \cdot \log(P)\right)$, где $N = P^{1+1/k}$.

1. Введение

В данной статье описывается новый параллельный алгоритм сортировки N элементов на P процессорах, где $N \geq P$. Этот алгоритм может быть эффективно реализован на широком круге параллельных компьютеров, включая гиперкуб, КОТА и кубическую сеть с циклами. В частности, алгоритм работает за время $O((N \log(N))/P)$ на любой из перечисленных архитектур в том случае, если $N = P^{1+1/k}$ для любого положительного значения константы k. Это первый алгоритм сортировки для перечисленных выше архитектур, в котором достигается такая производительность. Вдобавок, алгоритм сортировки будет расширен для достижения эффективной имитации Priority-CRCW PRAM используя гиперкуб, обменно-тасовочную или кубическую сеть с циклами. В оставшейся части данной главы проводится обзор различных моделей параллельных компьютеров и прошлых работ по параллельным алгоритмам сортировки.

---

[1] Перевод с анг. cubesort (здесь и далее прим. переводчика).
[2] Здесь и далее сокращение КС - кубическая сортировка.
[3] Перевод с анг. shuffle-exchange.
[4] Здесь и далее сокращение КОТА - компьютер с обменно-тасовочной архитектурой.
[5] Перевод с анг. cube-connected cycles.
[6] Priority-CRCW PRAM - Priority Concurrent Read Concurrent Write Parallel Random Access Machine – Параллельная Машина со Случайным Доступом и Приоритетными операциями Конкурирующего Чтения и Конкурирующей Записи.

Модели параллельных компьютеров, которые будут использованы в данной статье: PRAM [5], гиперкуб [10], обменно-тасовочная [10] и кубическая сеть с циклами [11]. Эти модели работают в ОКМД-режиме, при котором все процессоры исполняют одну и ту же инструкцию в любой момент времени. Модель PRAM – это модель с разделяемой памятью, в которой каждый процессор может обращаться к общей памяти в некоторую единицу времени. Priority-CRCW PRAM позволяет множеству процессоров читать «из» и записывать данные «в» одну и ту же ячейку памяти одновременно. В случае одновременной записи в одну и ту же ячейку, запись разрешается процессору с наименьшим номером из всех процессоров, которые пытаются выполнить запись в эту ячейку.

Модели гиперкуб, обменно-тасовочная и кубическая сеть с циклами состоят из множества процессоров, каждый из которых содержит локальную память и которые общаются друг с другом по соединительной сети. В гиперкубе, P процессоров нумеруются как $[0,\ P-1]$ и процессоры i и j соединяются, если двоичное представление i и j отличаются только в одном разряде. В обменно-тасовочной модели, P процессоров нумеруются как $[0,\ P-1]$ и процессоры i и j соединяются, если $j = Shuffle(i,P)$, $j = Unshuffle(i,P)$ или $j = Exchange(i)$, где $Shuffle(i,P) = 2i\ mod\ (P-1)$, $Unshuffle(i,P) = j$ если и только если $Shuffle(j,P) = i$, а $Exchange(i) = i + 1 - 2(i\ mod\ 2)$. Кубическая сеть с циклами состоит из P процессоров, причем $P = 2^K$, где $K = R + 2^R$. Процессоры пронумерованы парами (b,c), где b – это $(K-R)$-битное число, а c – R-битное число. Процессор (b,c) соединяется с процессором (d,e), если $\big((b=d) \wedge (c=e+1)\big) \vee \big((b=d) \wedge (c=e-1)\big)$, или если $(c=e)$, а b и d отличаются только одним битом в разряде c. Кубическая сеть с циклами и обменно-тасовочная модели являются практически реализуемыми, так как каждый процессор соединен только с фиксированным количеством других процессоров.

Один из самых ранних результатов по алгоритмам параллельной сортировки был получен Бэтчером. В работе [3] Бэтчер описал алгоритм битонической сортировки[7]. В работе [12] Стон показал, что битоническая сортировка может быть реализована на обменно-тасовочной модели. Что, в свою очередь, позволяет для этой модели отсортировать $N = P$ элементов за время $O(\log^2 N)$.

В работе [4] Боде и Стивенсон показали, что любой параллельный алгоритм сортировки для N элементов, основанный на сравнениях и обменах и использующий $P = N$ процессоров, может быть адаптирован под алгоритм сортировки N элементов, используя $P < N$ процессоров. Применяя их метод к битонической сортировке на обменно-тасовочной модели, они получили алгоритм с временной сложностью $O\big((N/P)\cdot\log(N/P) + (N/P)\cdot\log^2 P\big)$, работающий на $P \le N$ процессорах. Этот алгоритм дает выигрыш в скорости по сравнению с последовательным алгоритмом сортировки, основанным на сравнениях, только если $P \in O\left(2^{\sqrt{\log N}}\right)$.

В работе [6] Готлиб и Крускал предложили алгоритм для частного случая задачи сортировки. Они предложили обменно-тасовочный алгоритм для задачи о перестановках, в которой необходимо отсортировать N неповторяющихся элементов i, где $i \in \mathbb{N}$ и $i = [1,N]$. Их алгоритм для своей работы требует время $O\big(P^{9/2} + (N/P)\log P\big)$ и дает выигрыш в скорости по сравнению с последовательным алгоритмом сортировки, основанным на сравнениях, только в случае, если $P \in O\big((N\log N)^{2/9}\big)$. В своей работе Готлиб и Крускал утверждают, что им неизвестен оптимальный алгоритм для задачи о перестановках для случая, когда $P \notin O\big((N\log N)^{2/9}\big)$. Таким образом, данная статья улучшает результат, достигнутый Готлибом и Крускалом, в двух направлениях. Во-первых, алгоритм, представленный в данной работе, решает общую задачу сортировки, а не её частный случай. Во-вторых – дает увеличение в производительности, когда $N = P^{1+1/k}$ для любого положительного значения константы k.

[7] Калька с анг. bitonic sorting algorithm.

Прорывные результаты для задачи параллельной сортировки были достигнуты Аджитаем, Комлосом и Семереди [2]. Они создали сеть для сортировки N элементов, которая состоит из $O(N\log(N))$ сравнителей и глубиной $O(\log(N))$. Эта сеть была использована Лейтоном для создания практически реализуемой параллельной машины, которая сортирует N элементов за время $O(\log(N))$, если количество процессоров $P = N$ [8].

К сожалению, подход Лейтона имеет два серьёзных недостатка. Во-первых, он работает медленно для $P < 10^{100}$. Напротив, алгоритм, представленный в данной работе обладает гораздо меньшим значением константы и гораздо более вероятно его практическое применение. Во-вторых, сеть Лейтона не способна решать другие задачи, отличные от сортировки. Напротив, обменно-тасовочная сеть и кубическая сеть с циклами способны решать широкий круг задач.

Еще один важный результат, связанный с задачей параллельной сортировки был получен также Лейтоном. Лейтон недавно показал, что его алгоритм «колоночной сортировки»[8] [8] может быть использован в задаче сортировки $N = P^{1+1/k}$ элементов на P-процессорной обменно-тасовочной сети [9]. Он предоставил алгоритм с временной сложностью $O\left(k^T P^{1/k}\log P\right)$, где $T = 1/\log_4 1.5$ ($T \approx 3.419$). Алгоритм основывается на вызове колоночной сортировки по принципу вложенности так, что N элементов сортируется путем повторяющейся сортировки групп элементов, каждая из которых размером $P^{1/k}$. Кроме того, существует возможность сделать значение T меньше 1, используя идею Лейтона о «ближайшей сортировке»[9] [8,9]. Наконец, похожий результат использования колоночной сортировки был получен Аггарвалем [1]. Очевидна необходимость в большем количестве исследований по приложениям колоночной сортировки.

Ниже, статья делиться следующим образом. В Главе 2 содержится абстрактное описание нового алгоритма сортировки и доказательство его корректности. В Главе 3 показано, как этот алгоритм сортировки может быть эффективно реализован на различных параллельных компьютерах, а также представлен алгоритм имитации Priority-CRCW PRAM на обменно-тасовочной сети. На всем протяжении статьи будем считать, что N – это количество элементов для сортировки, а P – количеством доступных процессоров.

2. Кубическая сортировка

Данная глава содержит описание предлагаемого алгоритма сортировки, который авторы называют «кубической сортировкой». Описание КС, приведенное в данной главе, не зависит от вычислительной архитектуры, используемой для её реализации. КС работает путем повторяющегося деления N элементов на небольшие группы и сортируя каждую группу отдельно и параллельно с другими. В частности, пусть $N = M^D$, где M и D целые числа. Каждый шаг кубической сортировки разделяет M^D элементов, либо на M^{D-1} групп по M элементов в каждой, либо на M^{D-2} групп по M^2 элементов в каждой и сортирует группы параллельно.

M^D элементов для сортировки могут рассматриваться как D-мерный куб, каждая сторона которого длины M. Каждое местонахождение L в таком кубе описывается адресом $L = (L_D, L_{D-1}, \dots, L_1)$, где $(L_D, L_{D-1}, \dots, L_1)$ – это число в системе счисления с основанием M, состоящее из D цифр, а L_i – это проекция местонахождения L на i-ое измерение. Такая нумерация местонахождений в кубе соответствует порядку местонахождений, который будем называть *порядком главного ряда*[10]. Кубическая сортировка будет сортировать элементы в кубе в порядке главного ряда.

[8] Перевод с анг. columnsort.
[9] Перевод с анг. closesorting.
[10] Перевод с анг. row-major order.

В дополнение к представлению элементов в виде D-мерного куба, элементы можно рассматривать как множество кубиков меньшей размерности. *j-куб*, где $0 \leq j \leq D$, представляет собой множество M^j элементов с M-ричными адресами, которые различаются только в j младших разрядах. То есть, $A = (A_D, A_{D-1}, \dots, A_1)$ и $B = (B_D, B_{D-1}, \dots, B_1)$ попадают в один и тот же j-куб, тогда и только тогда, когда $\forall i(A_i = B_i) : j + 1 \leq i \leq D$. Каждый j-куб, где $0 \leq j \leq D$, может быть *чётным* или *нечётным*. j-куб, где $0 \leq j \leq D - 1$, чётный, если он содержит местонахождение L, для которого $L_{j+1}\ mod\ 2 = 0$, а иначе он нечётный. D-куб, который содержит все N элементов считается чётным.

Существует D различных разделений, обозначаемых как P_j, где $1 \leq j \leq D$, которые используются кубической сортировкой. Группа, принадлежащая разделению P_j состоит из множества элементов с M-ричными адресами, которые различаются только разрядами j и $j - 1$. Следует заметить, что каждая группа в P_1 содержит M элементов, в том время как каждая группа в оставшихся разделениях содержит M^2 элементов.

Наконец, иногда оказывается полезным рассматривать элементы j-куба как формирующими двумерный массив. *j-массив*, где $2 \leq j \leq D$, это $M^2 \times M^{j-2}$ массив элементов в j-кубе, в котором элементы в массиве расположены в порядке главной строки. Так, каждый $(j - 2)$-куб составляет ряд в j-массиве, а каждый $(j - 1)$-куб составляет собрание из M последовательных рядов в j-массиве. Также, каждый столбец j-массива является группой в P_j.

Для того, чтобы кубическая сортировка работала корректно, предполагается, что $D \geq 3$ и $M \geq (D - 1)(D - 2)$. Кубическая сортировка использует две подпрограммы, Sort_Ascending и Sort_Mixed. Подпрограмма Sort_Ascending(i) сортирует группы в разделении P_i по возрастающему порядку главной строки. Подпрограмма Sort_Mixed(i, j) сортирует группы в разделении P_j в порядке возрастания, если они находятся в чётных i-кубах, и в порядке убывания, если они находятся в нечётных i-кубах. Кубическая сортировка начинается с инициализации переменных M и D и вызова подпрограммы Cubesort(D). Псевдо-код кубической сортировки приведен ниже.

```
Cubesort(S)                     /* Описание работы кубической сортировки */
integer S;
{
    if S = 3 then
    {
                                /* ФАЗА 1: */
        Limit_Dirty_Cubes(S);
                                /* ФАЗА 2: */
        Sort_Mixed(S-1, S-1);
                                /* ФАЗА 3: */
        Merge_Dirty_Cubes(S, S);
    }
    else
    {
                                /* ФАЗА 1: */
        Limit_Dirty_Cubes(S);
        Limit_Dirty_Cubes(S);
                                /* ФАЗА 2: */
        Cubesort(S-1);
                                /* ФАЗА 3: */
        Merge_Dirty_Cubes(S, S);
    }
}

Limit_Dirty_Cubes(S)
integer S;
{
    if S > 2 then
```

```
        Limit_Dirty_Cubes(S-1);
    Sort_Ascending(S);
}

Merge_Dirty_Cubes(S, T)
integer S, T;
{
    Sort_Mixed(S, T);
    if T > 2 then
        Merge_Dirty_Cubes(S, T-2);
```

Вызов Cubesort(S) сортирует каждый чётный S-куб в возрастающем порядке главной строки, а каждый нечётный S-куб – в убывающем порядке. Для того, чтобы доказать, что кубическая сортировка работает корректно, необходимо использовать принцип 0-1 [7], который гласит "если сеть с n входными линиями сортирует 2^n всех возможных последовательностей из 0 и 1 в порядке неубывания, то эта сеть отсортирует любую произвольную последовательность из n чисел в порядке неубывания". В соответствии с принципом 0-1, последующие рассуждения полагаются на то, что вход состоит полностью из 0 и 1.

Следующие определения понадобятся для доказательства корректности алгоритма. Множество элементов *грязное*, если оно содержит и 0 и 1, в противном случае оно *чистое*. Последовательность из 0 и 1 называется *возрастающей*, если она вида 0^a1^b, где $a, b \geq 0$. Последовательность из 0 и 1 называется *убывающей*, если она вида 1^a0^b, где $a, b \geq 0$. Последовательность называется *монотонной*, если она или возрастающая, или убывающая. Последовательность из 0 и 1 называется *битонической*, если она вида $0^a1^b0^c$ или $1^a0^b1^c$ $a, b, c \geq 0$. j-массива называется *пересеченно-отсортированным*[11], если все его строки монотонны и если он содержит не более, чем 1 возрастающую грязную строку и не более, чем 1 убывающую строку. j-массива называется *полу-отсортированным*[12], если все его строки битонические и если он содержит не более, чем 1 грязную строку. j-массива называется *блочно-отсортированным*[13] *в порядке возрастания* (*убывания*), если он состоит из строк A, состоящих только из 0 (или 1), за которыми следуют грязные строки B, за которыми следую строки C, состоящие только их 0 (или 1), где $A, B, C \geq 0$. j-куб нызывается пересеченно-отсортированным (или полу-отсортированным, или блочно-отсортированным), если соответствующий ему j-массив является пересеченно-отсортированным (или полу-отсортированным, или блочно-отсортированным).

Далее доказывается корректность Cubesort(S). Доказательства были опущены с целью сохранить место для статьи.

ЛЕММА 1: Если j-массив изначально содержит B грязных строк и если колонки j-массива сортируются в порядке возрастания (убывания), то результирующий j-массив будет блочно-отсортирован в порядке возрастания (убывания) и будет содержать не более, чем B грязных строк.

ЛЕММА 2: После вызова $Limit\_Dirty\_Cubes(i)$, где $i \geq 2$, существует не более, чем $i - 1$ $(i-1)$-кубов в каждом i-кубе и грязные $(i-1)$-кубы последовательно расположены в каждом i-массиве.

ЛЕММА 3: Если массив изначально полу-отсортирован или пересеченно-отсортирован и если столбцы j-массива сортируются в порядке возрастания (убывания), то результирующий j-массив будет полу-отсортирован и блочно-отсортирован и порядке возрастания (убывания).

---

[11] Перевод с анг. cross-sorted.
[12] Перевод с анг. semi-sorted.
[13] Перевод с анг. block-sorted.

ТЕОРЕМА 1: После вызова $\mathrm{Cubesort}(3)$ каждый чётный 3-куб отсортируется в возрастающем порядке главной строки и каждый нечётный 3-куб отсортируется в убывающем порядке главной строки.

ЛЕММА 4: Когда $S > 3$, после первой фазы существует не больше 2-х грязных $(S-1)$-кубов в каждом S-кубе и эти грязные кубы смежный друг с другом в S-массиве.

ЛЕММА 5: Для любых значений S и T, где $1 \leq T \leq S \leq D$, если изначально каждый T-куб или полу-отсортирован, или пересеченно-отсортирован и если вызывается подпрограмма $Merge\_Dirty\_Cubes(S,T)$, то результирующие T-кубы все будут отсортированы. Более того, T-кубы, которые находятся в чётных S-кубах, будут отсортированы в порядке возрастания, а T-кубы, которые находятся в нечётных S-кубах, будут отсортированы в порядке убывания.

ТЕОРЕМА 2: $\mathrm{Cubesort}(S)$, где $3 \leq S \leq D$, сортирует каждый чётный S-куб в порядке вохрастания, а каждый нечётный S-куб в порядке убывания.

3. Реализация Кубической сортировки

Алгоритм кубической сортировки, описанный в предыдущей главе, сортирует $N = M^D$ элементов за $O(D^2)$ этапов, при этом каждый этап состоит из параллельной сортировки групп, содержащих $O(M^2)$ элементов. В этой главе будет показано, как кубическая сортировка может быть выполнена на различных параллельных моделях.

Сперва будет представлена реализация кубической сортировки для обменно-тасовочной сети. Будем считать, что N – это количество элементов для сортировки, а P – количеством доступных процессоров, причём $N = P^{1+1/k}$. Элементы, предназначенные для сортировки, сортируются в массиве A размера N, где A_i находится на процессоре $j = \lfloor i/P^{1/k} \rfloor$, для $0 \leq i \leq N-1$. Для того, чтобы использовать алгоритм, описанный в предыдущей главе, положим $D = 2k+2$, $M = P^{1/k}$, а также, что местонахождение $L = (L_D, L_{D-1}, \dots, L_1)$ в D-мерном кубе соответствует A_L.

Перед тем, как группы разделения будут отсортированы, данные перестраиваются так, чтобы каждая группа находилась в пределах одного процессора. Для того, чтобы выполнить такое перестроение, используется 2 вида перестановок $M\text{-}Shuffle$ и $M\text{-}Unshuffle$. $M\text{-}Shuffle$ для N элементов определяется как $M\text{-}Shuffle(X,N) = MX \bmod (N-1)$. Перестановка $M\text{-}Unshuffle$ обратна перестановке $M\text{-}Shuffle$, так что, $M\text{-}Unshuffle(Z,N) = X \; iff \; M\text{-}Shuffle(X,N) = Z$.

$P^{1/k}$ элементов, которые расположены локально к каждому процессору могут быть отсортированы за время $O\left((1/k) \cdot P^{1/k} \cdot \log P\right)$. Также, $M\text{-}Shuffle$ и $M\text{-}Unshuffle$ для элементов, подлежащих сортировке, могут быть выполнены каждая за время $O\left((1/k) \cdot P^{1/k} \cdot \log P\right)$. В связи с тем, что реализация кубической сортировки для обменно-тассовочной сети состоит из $O(D^2) = O(k^2)$ независимых процессов сортировки, локальных по отношению к процессорам, и $O(k^2)$ процессов, вычисляющих функции $M\text{-}Shuffle$ и $M\text{-}Unshuffle$, общее время работы алгоритма оценивается как $O\left(k \cdot P^{1/k} \cdot \log P\right)$.

Реализация кубической сортировки для гиперкуба и кубической сети с циклами схожа с реализацией для обменно-тасовочной сети. Из-за ограниченности места будет приведен только результат такой реализации. Кубическая сортировка может быть реализована за время $O\left(k^2 \, P^{1/k} \, \log P\right)$ для моделей гиперкуб и кубическая сеть с циклами.

В приведенных рассуждениях предполагалось, что $N = P^{1+1/k}$. Однако, кубическая сортировка может быть использована в том случае, когда число элементов в расчете на процессор растет не так быстро. В частности, когда $N = P \log P$ кубическая сортировка работает за время

$O(\log^3 P/\log\log P)$ на P-процессорной обменно-тасовочной сети. Снова, ограниченность места препятствует включению этого алгоритма.

Наконец, кубическая сортировка может быть использована для имитации Priority-CRCW PRAM на компьютере с обменно-тасовочной архитектурой. Из-за ограниченности места приведен будет только результат. Элементарная операция Priority-CRCW PRAM, состоящей из N процессоров и N блоков памяти, может быть реализована за время $O\left(k \cdot P^{1/k} \cdot \log P\right)$ на P-процессорной обменно-тасовочной сети, где $N = P^{1+1/k}$.

Благодарности

Авторы хотели бы выразить благодарность проф. Т. Лейтону за обращение их внимания на его расширения колоночной сортировки для оптимальной сортировки, а также за множество полезных дискуссий. Также, авторы чувствуют себя в долгу перед проф. S. Hambrusch, док. M. Snir, проф. L. Snyder и док. E. Upfal за их полезные комментарии по содержанию данной статьи. Работа R. Cypher была поддержана грантом NSF в рамках аспирантских программ.

Библиографический указатель

1. A. Aggarwal, Unpublished manuscript, 1986.
2. M. Ajtai, J. Komlos, E. Szemeredi, "An $O(n \log n)$ Sorting Network", Proc. 15th Annual Symposium on Theory of Computing, 1983, pp. 1-9.
3. K.E. Batcher, "Sorting Networks and their Applications", 1968 AFIPS Conference Proceedings, pp. 307-314.
4. G. Baudet, D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers", IEEE Transactions on Computers, vol. c-27, no. 1, January 1978, pp. 84-87.
5. A. Borodin, J.E. Hopcroft, "Routing, Merging and Sorting on ParalM Models of Computation", Proc. 14th Annual Symposium on Theory of Computing, 1982, pp. 338-344.
6. A. Gottlieb, C.P. Kruskal, "Complexity Results for Permuting Data and Other Computations on Parallel Processors", Journal of the ACM, vol. 31, no. 2, April 1984, pp. 193-209.
7. D.E. Knuth, "The Art of Computer Programming, Vol. 3: Sorting and Searching", Addison-Wesley, Reading, MA, 1973.
8. T. Leighton, "Tight Bounds on the Complexity of ParalM Sorting", IEEE Transactions on Computers, vol. c-34, no. 4, April 1985, pp. 344-354.
9. T. Leighton, Personal communication.
10. D. Nassimi, S. Sahni, "Data Broadcasting in SIMD Computers", IEEE Transactions on Computers, vol. c-30, no. 2, February 1981, pp. 101-107.
11. F.P. Preparata, J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation", Communications of the ACM, vol. 24, no. 5, May 1981, pp. 300-309.
12. H.S. Stone, "Parallel Processing with the Perfect Shuffle", IEEE Transactions on Computers, vol. c-20, no, 2, February 1971, pp. t53-161.