第七讲 排序算法二

大纲

- 基本概念和排序网络
- 双调排序
- 冒泡排序和它的变种
- 快速排序
- 桶和采样排序
- 其它排序算法

排序

- 最常用，发展成熟的计算”核心“
- 排序可以使基于比较的或基于非比较的。
- 基于比较的排序的基本操作是比较-交换操作（compare-exchange）
- 任何n 个元素的比较排序的下界是$\Theta(n\log n)$
- 我们主要关注基于比较的排序算法

排序基本概念和排序网络

排序基本概念

- 什么是并行排序的序列？输入和输出序列存放在哪里？
- 我们假设输入和输出序列式分布式的摆放的
- 排好序的序列应该按如下性质摆放：每个部分的序列是有序的；并且当$i < j$时，每个P_i'上的元素都小于P_j'的元素。

并行比较的交换操作

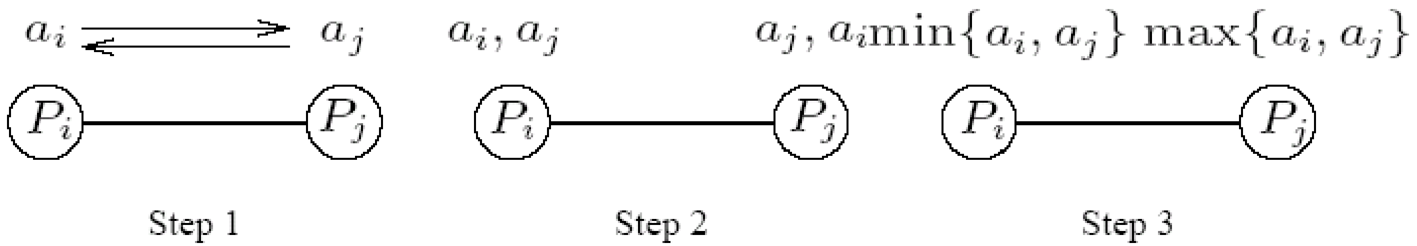


图1

一个并行-比较交换操作。进程P_i和P_j互相发送他们的元素给对方。进程P_i发保存min{a_i,a_j}，P_j保存max{a_i, a_j}

串行比较器、并行比较器

- 一个串行比较器在并行意义下应该是什么样的?
- 如果每个处理器有一个元素，那么id小的那个处理器在比较交换操作后保存较小的元素。这个操作可以在 $t_s + t_w$ 时间内完成
- 如果每个处理器的元素多于1.假设参与这个操作的处理器各自有 n/p 个元素
  - 两个进程将2n/p个元素排序，并重新分配：一个得到较小的n/p个元素，另一个得到较大的n/p个元素
  - 我们将这个操作称为比较分裂（compare split）。
- 完成比较分裂操作后。如果$i < j$.则，处理器 P_i 保存较小的 n/p 个元素，而处理器 P_j,保存较大的n/p 个元素
- 当两个局部的序列是局部有序时，一个比较分裂操作的通信耗时 $(t_s + t_w n/p)$

并行比较分裂操作

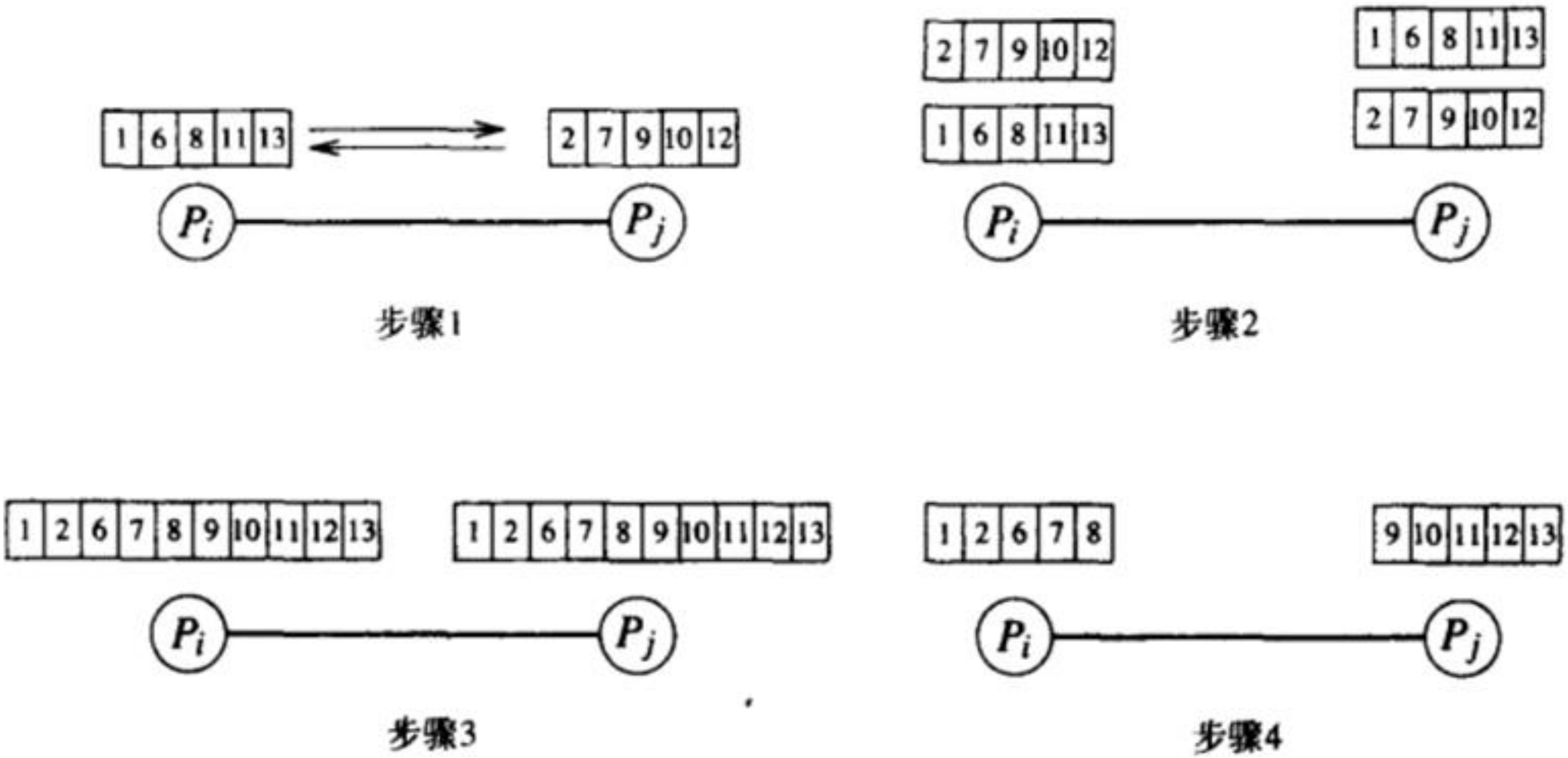
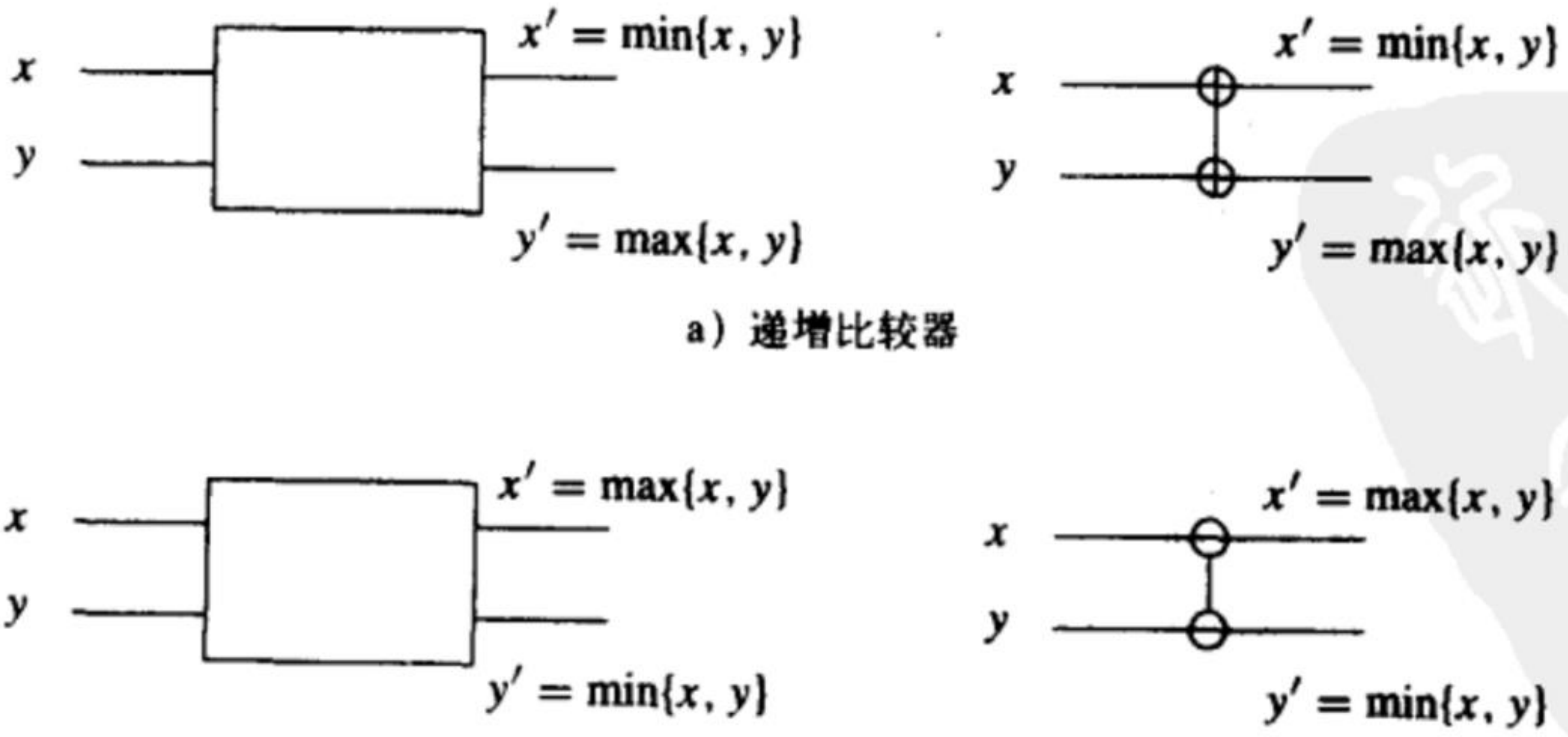


图2　比较分裂操作。每个进程发送n/p个元素的块到其他进程。每个进程将接收到的块与它们自己的块合并，只保留适合的那一半块。在这个例子中，进程P_i保留较小的那些元素，而进程P_j保留较大的那些元素

排序网络

- 比较器网络是专门为排序设计的
- 一个比较器是拥有两个输入x和y以及两个输出 x'和 y'的设备。对于一个递增比较器，$x' = \min\{x,y\}$ 和 $y' = \max\{x,y\}$，相反也是一样的。
- 我们将递增比较器记为⊕， 递减比较器为⊖
- 网络的速度与其深度成正比

排序网络：比较器



a）递增比较器

b）递减比较器

图3 比较器的图示

比较网络

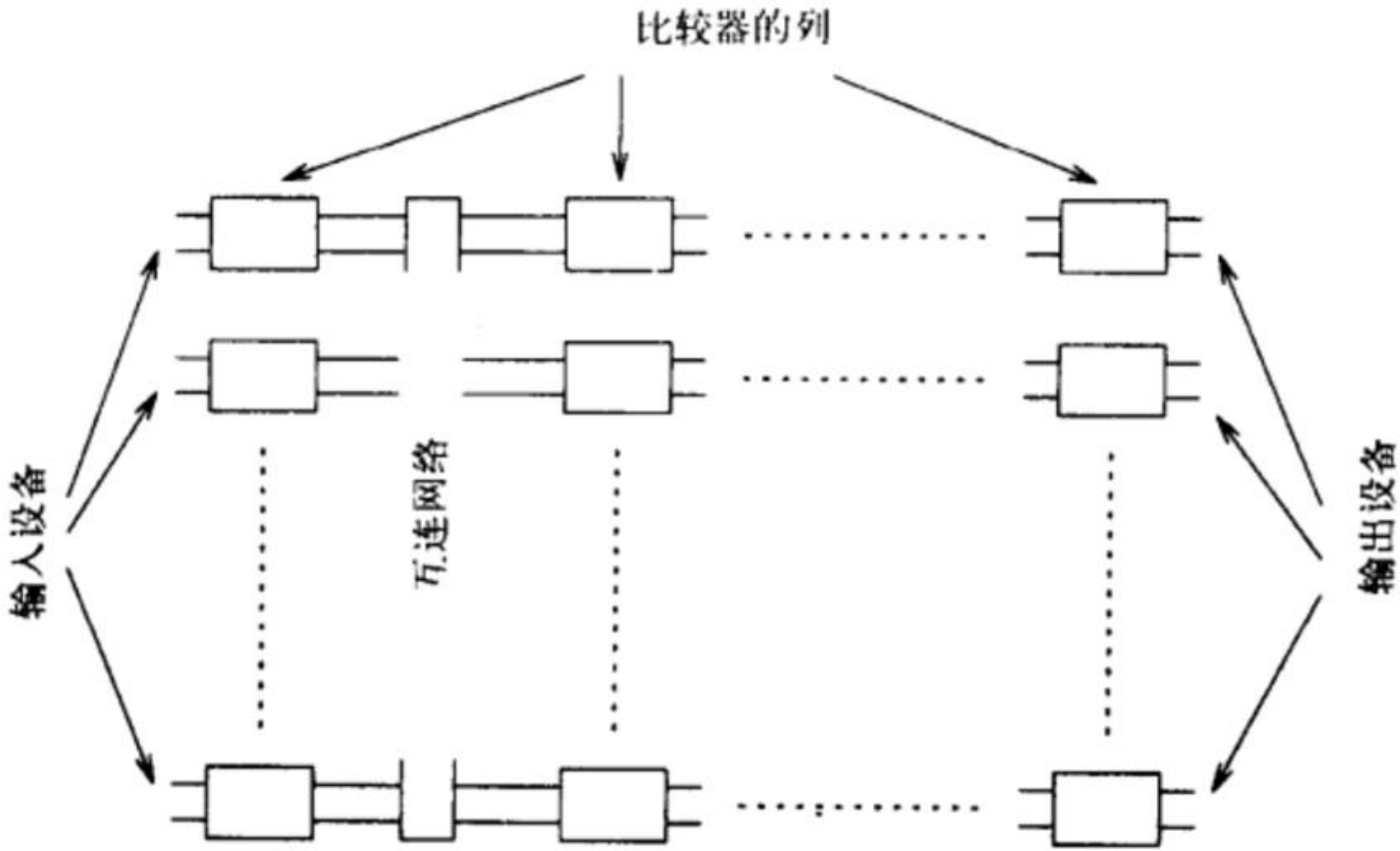


图4　一个典型的排序网络。每个排序网络由一连串的列组成，每个列包含许多并行连接的比较器

排序网络：双调排序

排序网络：双调排序

- 一个双调排序网络可以用 $\Theta(\log^2 n)$ 的时间对 n 个元素进行排序
- 一个双调序列有两种顺序- 先递增然后递减，反之亦然。通过循环移位可以得到上述双调序列的也是双调序列
- $\langle 1,2,4,7,6,0 \rangle$ 是一个双调序列，因为它先递增，然后递减。$\langle 8,9,2,1,0,4 \rangle$ 也是一个双调序列，因为它是$\langle 0,4,8,9,2,1 \rangle$的循环移位
- 双调排序网络的核心是将一个双调序列重排为一个有序序列

排序网络：双调排序

- 令$s = \langle a_0, a_1, \ldots, a_{n-1} \rangle$是双调序列，满足$a_0 \le a_1 \le \cdots \le a_{n/2-1}$ 和 $a_{n/2} \ge a_{n/2+1} \ge \cdots \ge a_{n-1}$.
- 考虑如下的s:的子列

$$s_1 = \langle \min\{a_0, a_{n/2}\}, \min\{a_1, a_{n/2+1}\}, \ldots, \min\{a_{n/2-1}, a_{n-1}\} \rangle$$
$$s_2 = \langle \max\{a_0, a_{n/2}\}, \max\{a_1, a_{n/2+1}\}, \ldots, \max\{a_{n/2-1}, a_{n-1}\} \rangle \tag{1}$$

- 注意 s_1 和s_2 都是双调序列，而且每个 s_1 的元素都小于s_2的。
- 我们可以将这个过程递归的应用到s_1 和s_2上，从而得到有序的序列

排序网络：双调排序

原始序列	3	5	8	9	10	12	14	20	95	90	60	40	35	23	18	0
第1次分裂	3	5	8	9	10	12	14	0	95	90	60	40	35	23	18	20
第2次分裂	3	5	8	0	10	12	14	9	35	23	18	20	95	90	60	40
第3次分裂	3	0	8	5	10	9	14	12	18	20	35	23	60	40	95	90
第4次分裂	0	3	5	8	9	10	12	14	18	20	23	35	40	60	90	95

图5 通过一系列log 16双调分裂合并一个有16个元素的双调序列

排序网络：双调排序

- 我们可以通过执行双调合并算法轻易地构造一个排序网络
- 这样的排序网络叫做双调合并网络
- 这个网络由 log n 列构成。每一列包含了 $n/2$ 个比较器并且执行双调合并中的一步。
- 我们将一个有n 个输入的双调合并网络记为⊕BM[n]。
- 用⊖ 比较器代替比较器 ⊕，可以得到一个递减的输出序列。记为 ⊖BM[n]。

排序网络：双调排序

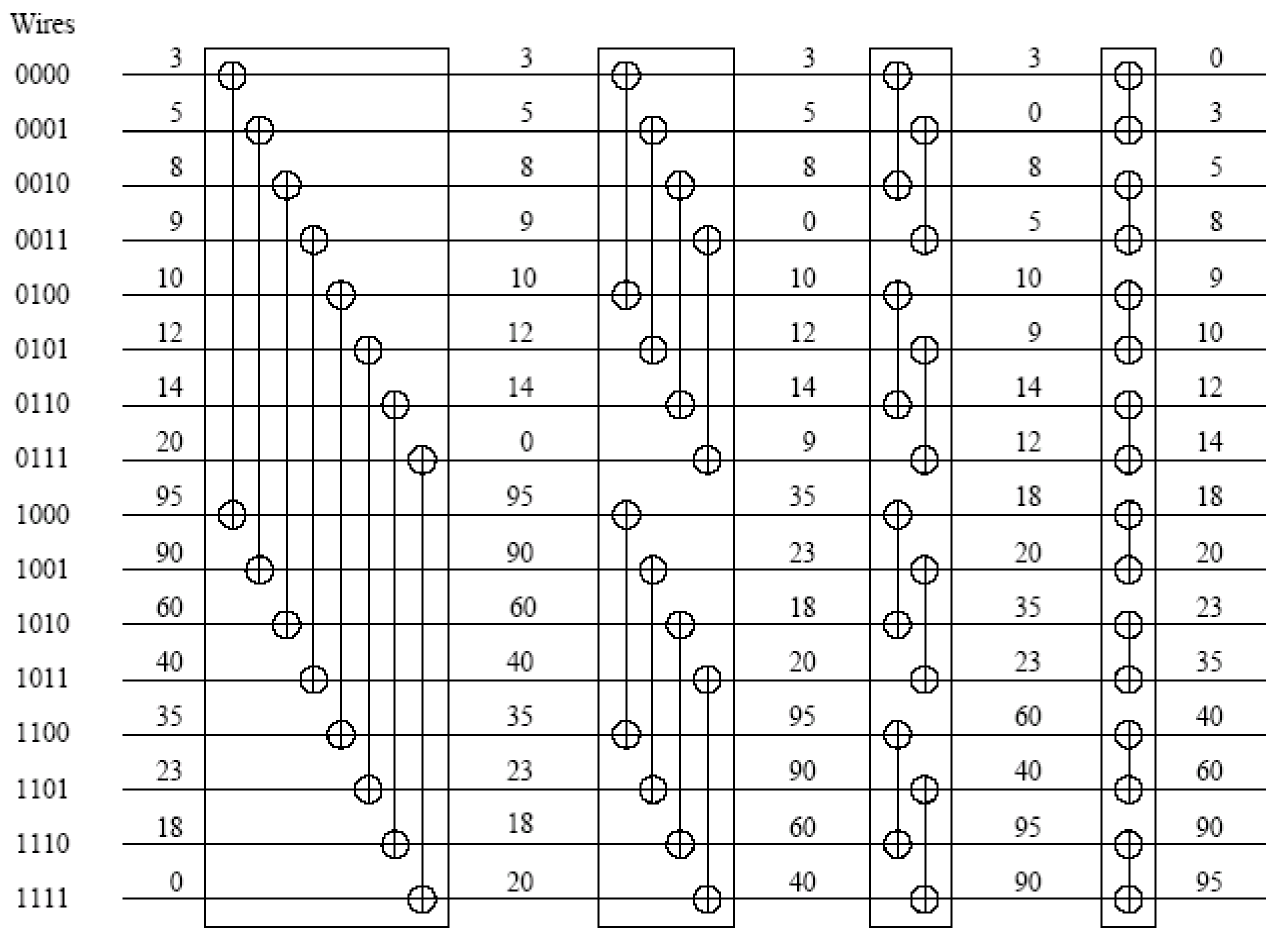


图6

⊕BM[16] 网络，输入时 0,1,..., n - 1,

第一列是输入的二进制形式。每一列比较器都用方框隔开

排序网络：双调排序

- 怎样用双调合并对一个无序的序列进行排序?
- 一个长为2的序列是双调序列。
- 一个长为4的双调序列可以这样构造：对前两个元素使用 $\oplus BM[2]$，对后两个使用 $\ominus BM[2]$。
- 重复这个过程可以生成更长的双调序列。

排序网络：双调排序

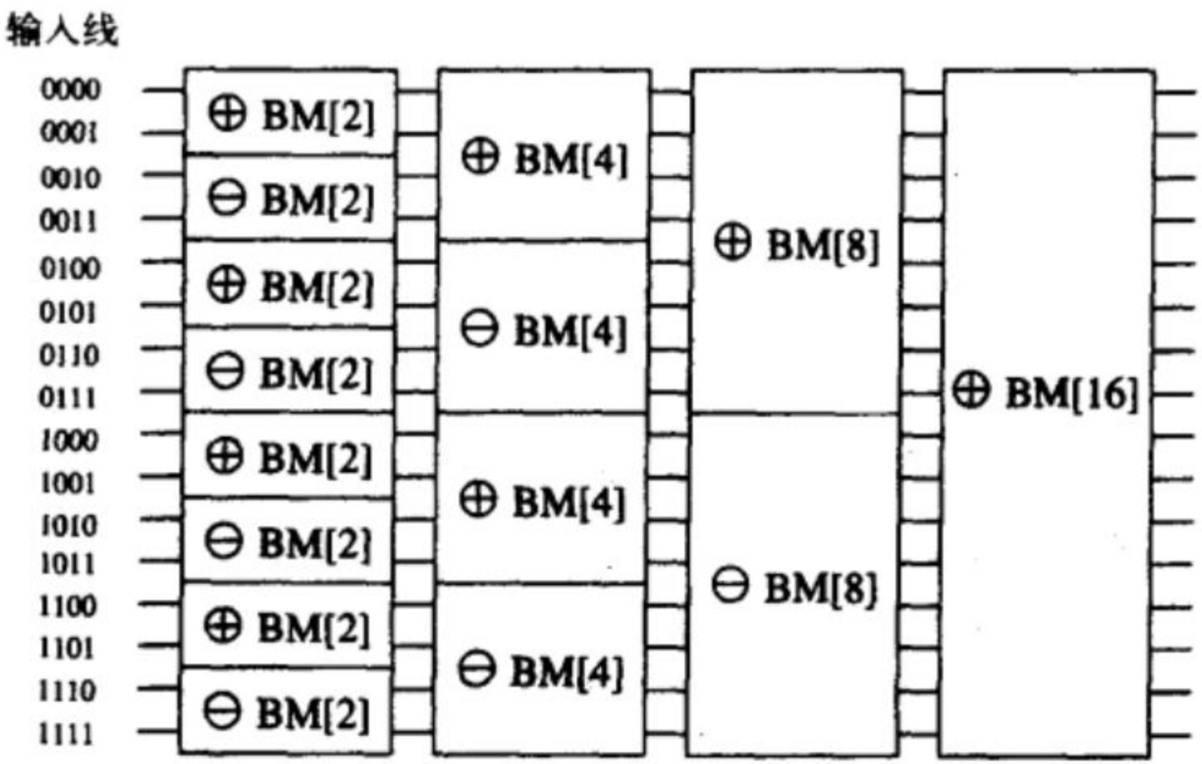


图7 将输入序列转化成双调序列的网络图示

注：在这个例子中，⊕BM[k]和⊖BM[k]表示输入大小为k的双调合并网络，分别用≈比较器和⊖比较器。最后的合并网络(⊕BM[16])对输入排序，在本例中n = 16。

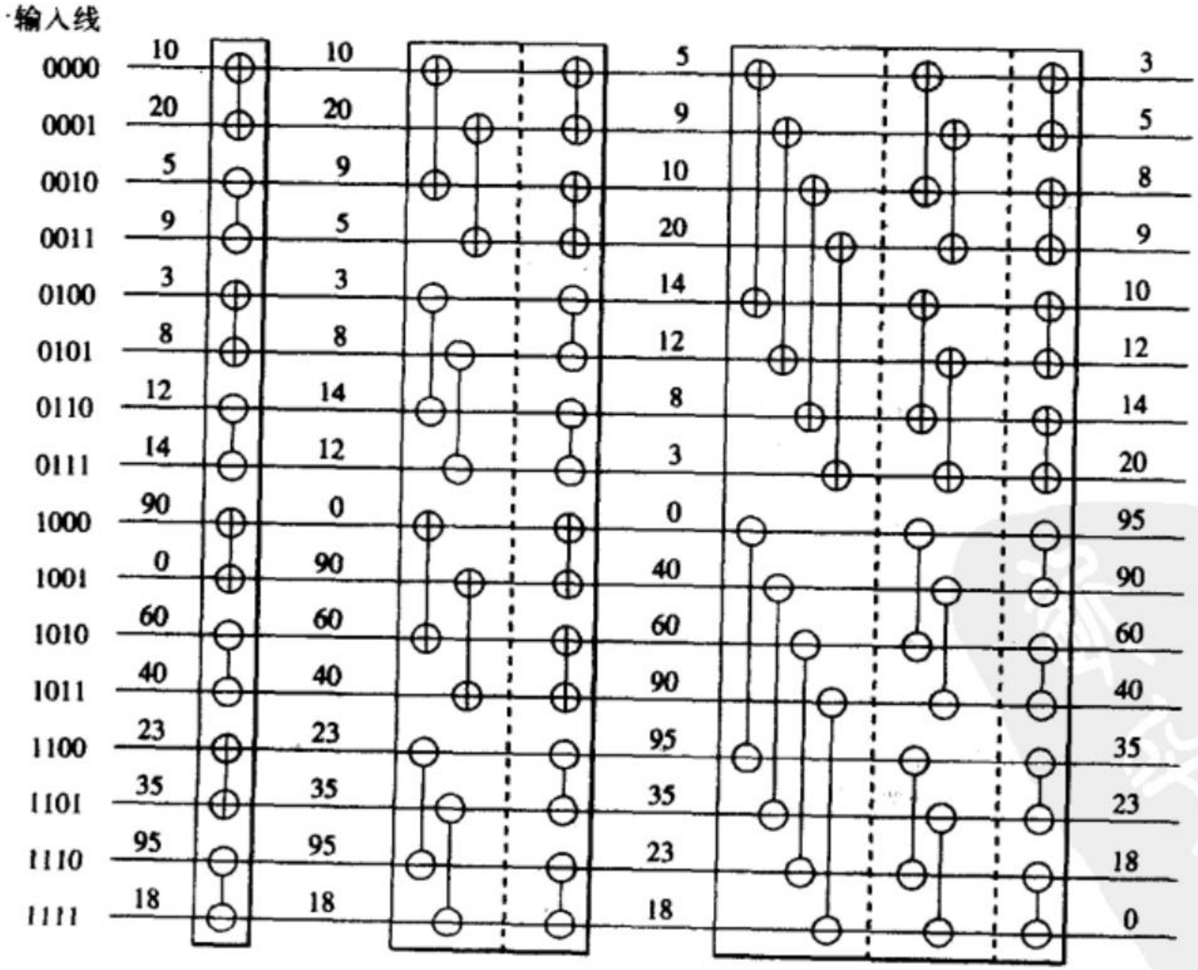


图8 将包含16个无序元素的输入序列转换成双调序列的比较网络

注：与图6相比，在每个双调合并网络上的比较器的列都画在一个单独的方框中，由虚线分隔开。

排序网络：双调排序

- 这个网络的深度为$\Theta(\log^2 n)$。
- 网络中的每个阶段都包含了 $n/2$ 个比较器。这个网络的串行执行复杂度为 $\Theta(n\log^2 n)$。

将双调排序映射到超立方上

- 考虑每个处理器一个元素的情况。问题变成了如何将双调排序中的线映射到超立方互连网络上。
- 注意我们原来的例子中，比较-交换操作总是在两个下标只有一个二进制位不同的线之间进行的。
- 这意味着，直接将线映射到进程上，所有的通讯都在邻居之间进行！

将双调排序映射到超立方上

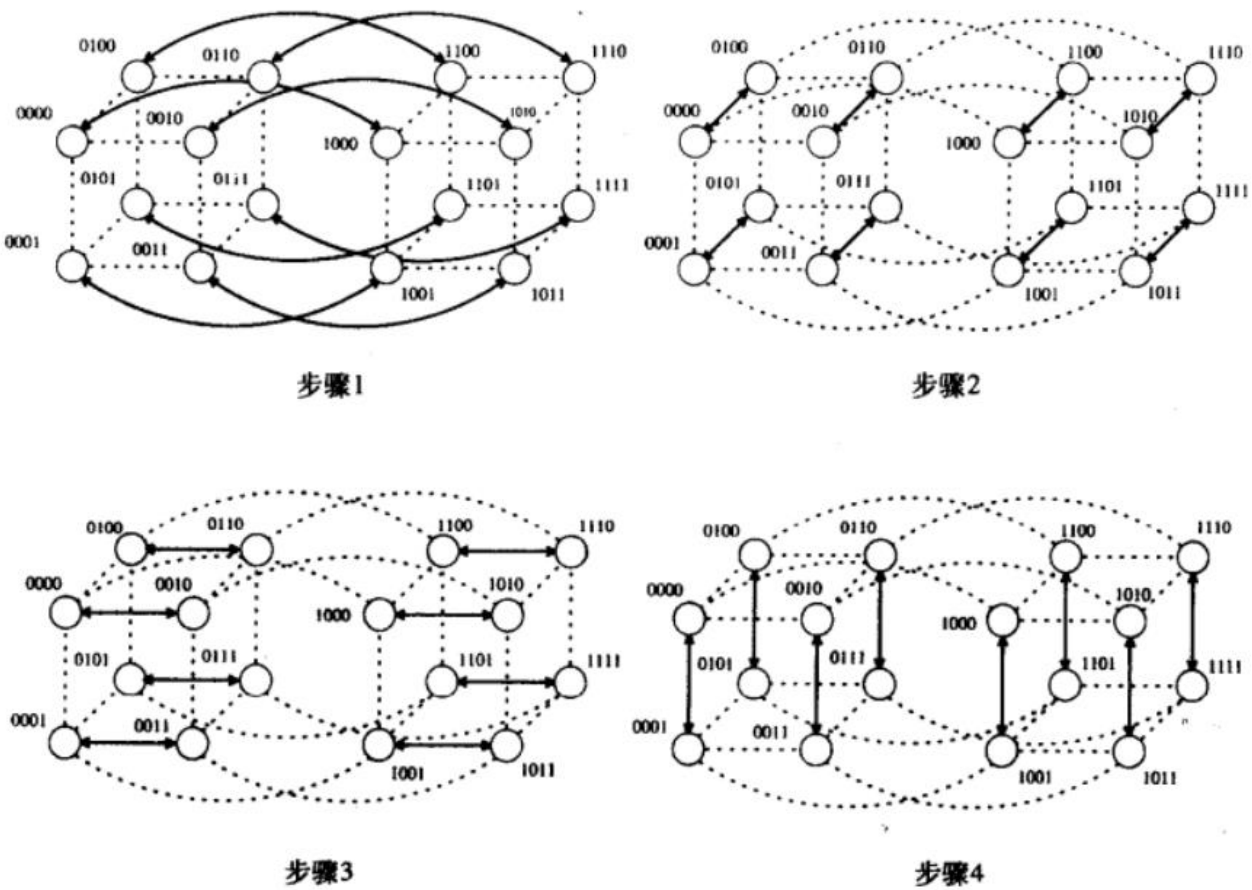


图 9 双调排序最后阶段的通信。每条线都被映射到一个超立方体进程；每个连接都代表进程间的一次比较-交换

将双调排序映射到超立方上

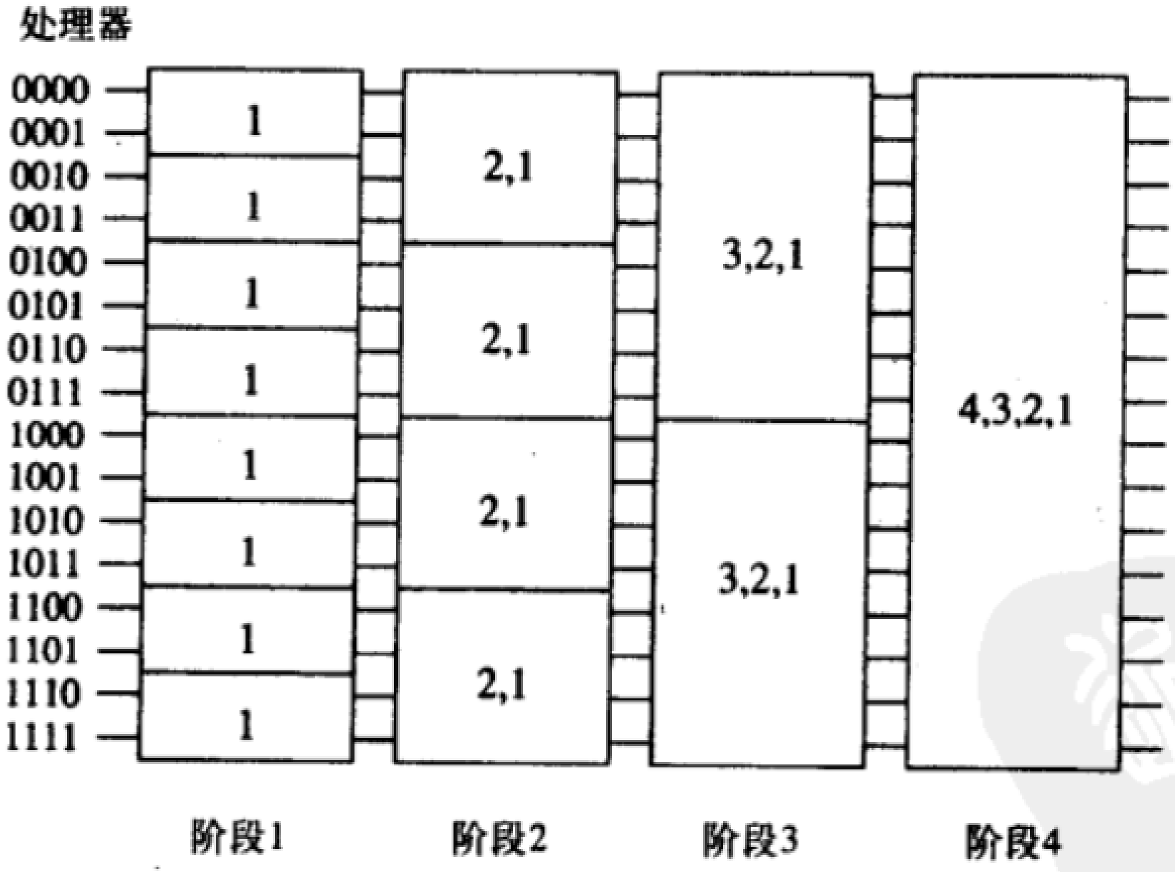


图 10　在超立方体上的双调排序的通信特征。在算法的每个阶段，进程通信沿着显示的维进行

将双调排序映射到超立方上

算法 1　在$n = 2^d$个进程的超立方体上的双调排序并行形式。在这个算法中，label是进程的标号，d是超立方体的维

```
procedure BITONIC_SORT(label, d)
begin
    for i := 0 to d − 1 do
        for j := i downto 0 do
            if (i + 1)st bit of label ≠ jth bit of label then
                comp_exchange_max(j);
            else
                comp_exchange_min(j);
end BITONIC_SORT
```

将双调排序映射到超立方上

- 在算法的每一步中，每个进程执行一个比较-交换操作（与最近的邻居通讯一个字）

- 由于每步只需要 $\Theta(1)$ 时间，则并行时间是

$$T_p = \Theta(\log^2 n) \qquad (2)$$

- 相对于它的串行版本，这个算法是成本最优的。但相对于最好的串行排序算法

将双调排序映射到格网上

- 一个格网的连通性比超立方差，所以在映射上会产生很多开销

- 考虑行为主的混洗映射

将双调排序映射到格网上

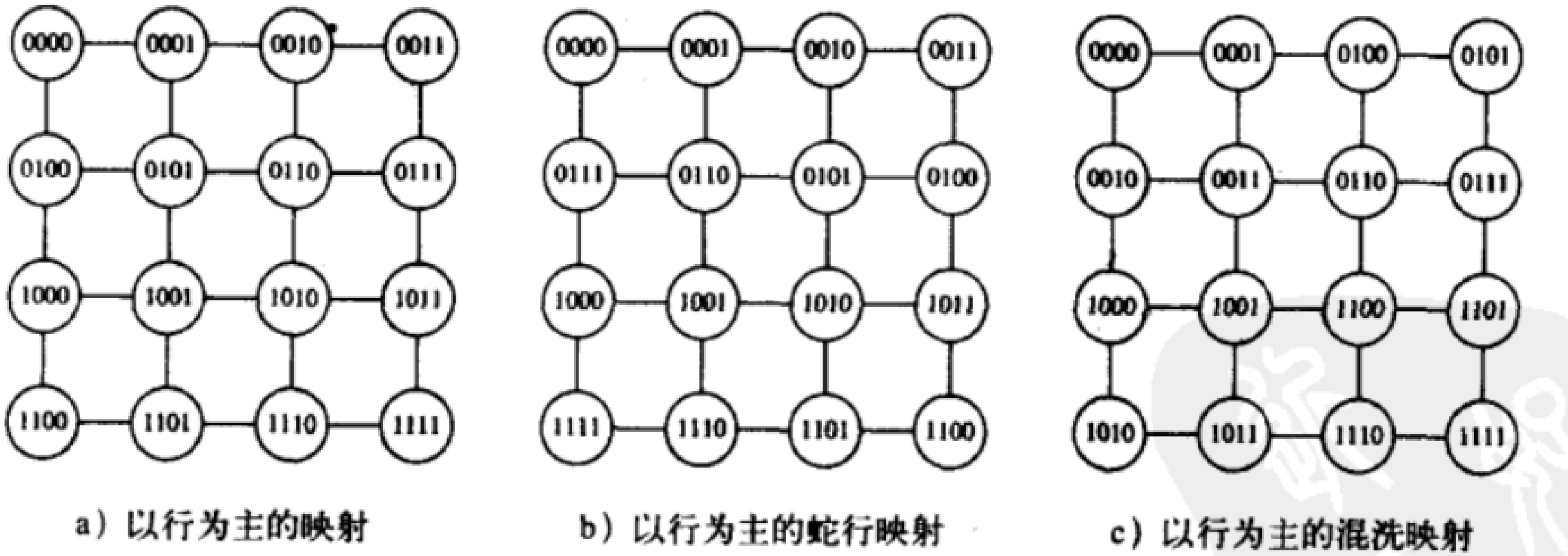


图11　映射双调排序网络的输入线到进程格网的不同方法

将双调排序映射到格网上

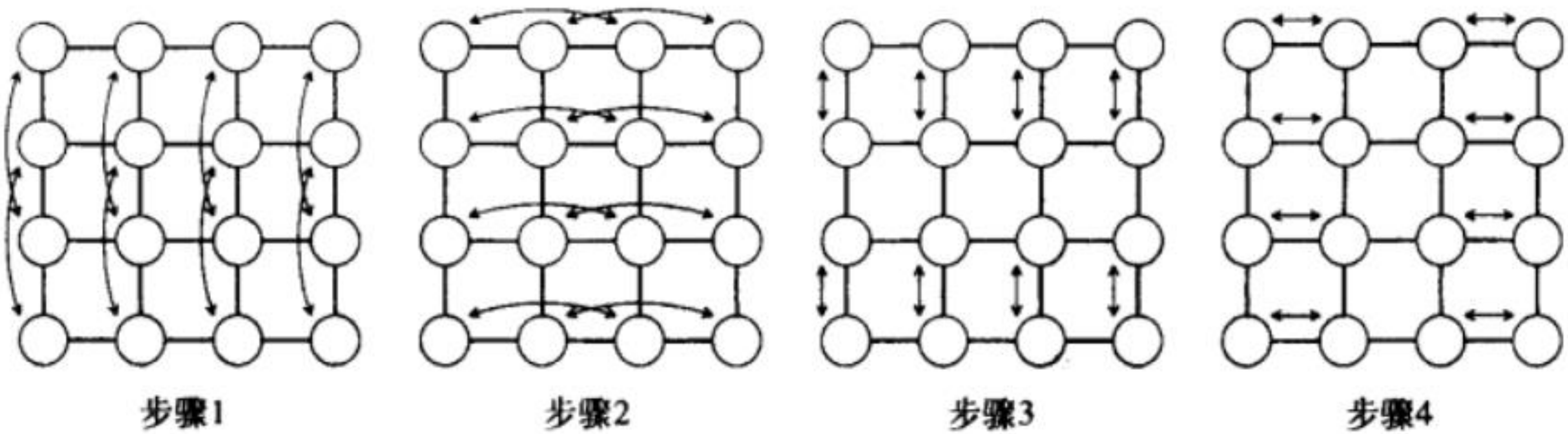


图12　格网上n =16的双调排序算法的最后一个阶段，使用以行为主的混洗映射。在每一步，进程对比较-交换它们的元素。箭头表明执行比较-交换操作的进程对

将双调排序映射到格网上

- 在以行为主的混洗映射中。低位-优先i^{th} 比特不同的线路上的进程，单个的通讯量为 $2^{\lfloor (i-1)/2 \rfloor}$
- 每个进程的总通讯量为 $\sum_{i=1}^{\log n}\sum_{j=1}^{i} 2^{\lfloor (j-1)/2 \rfloor} \approx 7\sqrt{n}$, or $\Theta(\sqrt{n})$

  每个进程的总计算量为 $\Theta(\log^2 n)$.
- 并行执行时间是:

$$T_P = \overbrace{\Theta(\log^2 n)}^{\text{comparisons}} + \overbrace{\Theta(\sqrt{n})}^{\text{communication}}.$$

- 这不是成本最优的。

每个进程一个元素块

- 每个进程负责一个包含 n/p个元素的块
- 第一步是本地块的本地排序
- 每个块之间的比较-交换操作被替换为比较-分裂操作
- 我们可以看到双调排序网络只需要$(1 + \log p)(\log p)/2$ 步，就可以高效的完成排序

每个进程一个元素块：超立方

- 一开始，进程耗时 $\Theta((n/p)\log(n/p))$ 对各自的 n/p 个元素排序，然后进行$\Theta(\log^2 p)$ 步的比较-分裂操作
- 这个形式的并行执行时间是：

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}^{\text{comparisons}} + \overbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}^{\text{communication}}.$$

- 与最优的串行排序相比，这个算法只能有效运用 $p = \Theta(2^{\sqrt{\log n}})$ 个处理器
- 由通讯和额外工作决定的等效率函数为 $\Theta(p^{\log p}\log^2 p)$.

每个进程一个元素块：格网

- 并行执行时间是

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}^{\text{comparisons}} + \overbrace{\Theta\left(\frac{n}{\sqrt{p}}\right)}^{\text{communication}}$$

- 这个形式可以有效的使用 $p = \Theta(\log^2 n)$ 个进程
- 等效率函数为 $\Theta(2^{\sqrt{p}}\sqrt{p})$.

并行双调排序的性能

表 1　在p个进程上对n个元素进行并行形式双调排序的性能

结　构	$E=\Theta(1)$时的最大进程数	对应的并行运行时间	等效率函数
超立方体	$\Theta(2^{\sqrt{\log n}})$	$\Theta(n/(2^{\sqrt{\log n}})\log n)$	$\Theta(p^{\log p}\log^2 p)$
格网	$\Theta(\log^2 n)$	$\Theta(n/\log n)$	$\Theta(2^{\sqrt{p}}\sqrt{p})$
环	$\Theta(\log n)$	$\Theta(n)$	$\Theta(2^p p)$

冒泡排序和它的变种

冒泡排序和它的变种

- 串行冒泡算法比较并交换序列中的相邻元素

procedure BUBBLE_SORT(n)
begin
    for $i := n - 1$ **downto** 1 **do**
        for $j := 1$ **to** i **do**
            compare-exchange(a_j, a_{j+1});
end BUBBLE_SORT

冒泡排序和它的变种

- 冒泡排序的复杂度为$\Theta(n^2)$
- 冒泡排序很难并行化，因为这个算法没有并发度
- 一个简单的变种，可以得到并发度——奇偶交换排序

奇偶交换排序

```
procedure ODD-EVEN(n)
begin
    for i := 1 to n do
    begin
        if i is odd then
            for j := 0 to n/2 - 1 do
                compare-exchange(a_{2j+1}, a_{2j+2});
        if i is even then
            for j := 1 to n/2 - 1 do
                compare-exchange(a_{2j}, a_{2j+1});
    end for
end ODD-EVEN
```

串行奇偶交换排序算法

奇偶交换排序

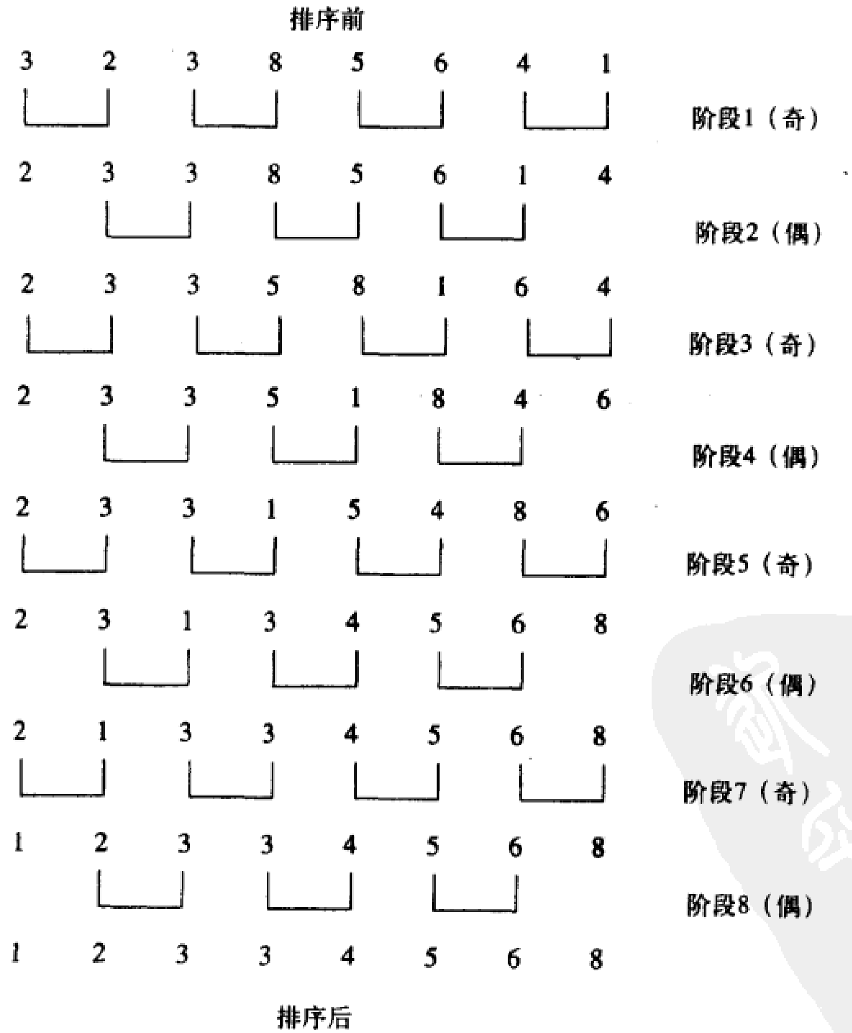


图 13　排序n = 8个元素，使用奇偶转换排序算法。
在每个阶段，比较n = 8个元素

奇偶交换排序

- 经过 n 步的奇偶互换后，序列就有序了
- 算法的每一步（奇或偶）都需要 $\Theta(n)$ 个比较器
- 串行复杂度是 $\Theta(n^2)$.

并行奇偶交换排序

- 考虑每个进程负责一个元素的情况
- 总共需要 n i步，每步中，每个处理器做一次比较-交换操作
- 该形式的并行执行时间为 $\Theta(n)$.

- 对于串行版本的奇偶排序，它是成本最优的，但对于最快的串行排序就不是了。

并行奇偶交换排序

```
procedure ODD-EVEN_PAR(n)
begin
    id := process's label
    for i := 1 to n do
    begin
        if i is odd then
            if id is odd then
                compare-exchange_min(id + 1);
            else
                compare-exchange_max(id − 1);
        if i is even then
            if id is even then
                compare-exchange_min(id + 1);
            else
                compare-exchange_max(id − 1);
    end for
end ODD-EVEN_PAR
```

并行奇偶交换排序

并行奇偶交换排序

- 考虑每个进程负责 *n/p* 个元素的块的情况
- 第一步是一个本地的排序
- 在接下来的步骤中，比较-交换操作被替换为比较分裂操作。
- 该形式的并行执行时间为：

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta(n)}^{\text{comparisons}} + \overbrace{\Theta(n)}^{\text{communication}}.$$

并行奇偶交换排序

- 当 $p = O(\log n)$时，该并行形式是成本最优的
- 该形式的等效率函数为$\Theta(p2^p)$.

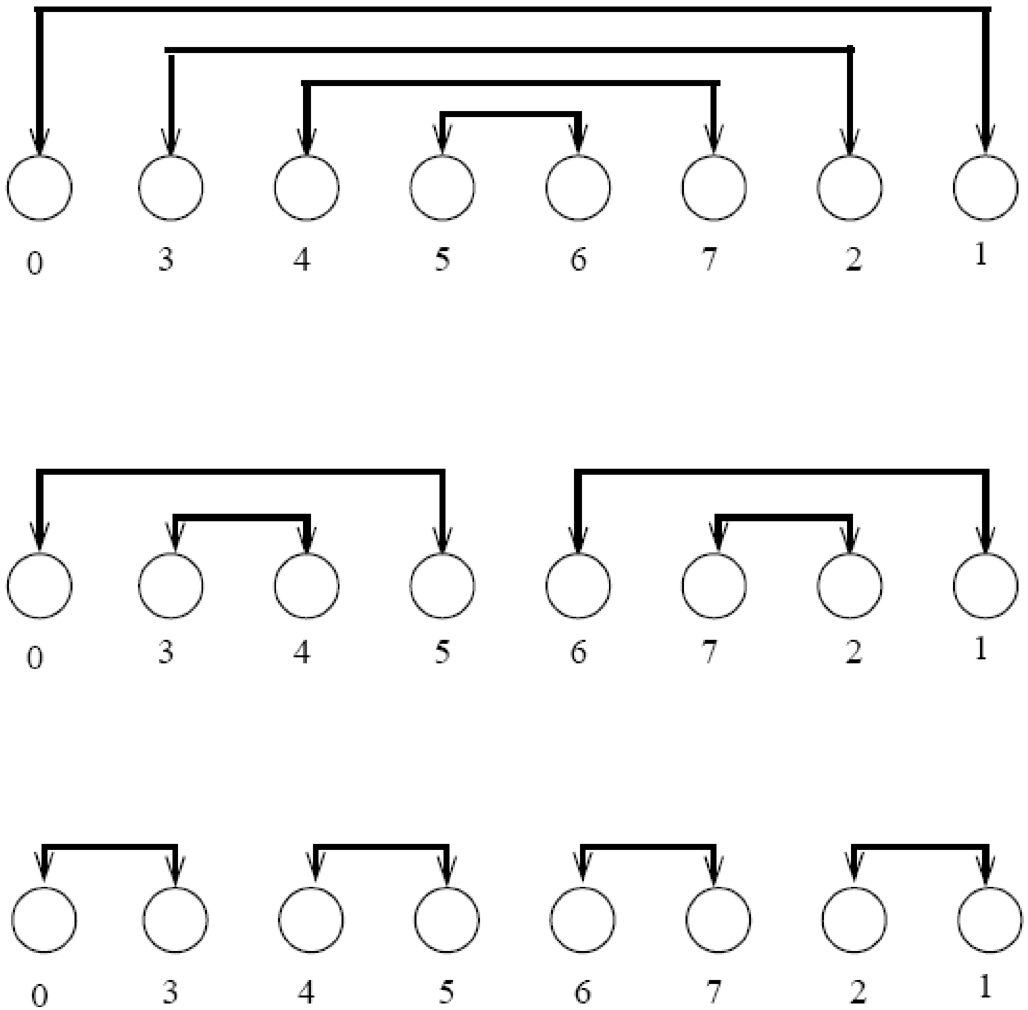
希尔排序

- 考虑用 p 个进程对 n 个元素进行排序的情况
- 第一阶段，数组中相聚远的进程互相进行比较-分裂操作。
- 第二阶段，算法变为一个奇偶转换排序

并行希尔排序

- 首先，每个进程在内部将 n/p 个元素排好序
- 然后，每个进程与它在数组中相反顺序对应的进程配对。即：P_i, 进程（$i < p/2$）与进程 P_{p-i-1} 配对。
- 配对的进程进行比较-分裂操作
- 进程被分为两个 $p/2$ 大小的组。接着在每个组里重复上述的步骤。

并行希尔排序



- 在一个8进程数组上的并行希尔排序的第一阶段的例子

并行希尔排序

- 每个进程需要进行$d = \log p$ 次比较-分裂操作
- 对于拥有 $O(p)$ 的对分带宽的网络，每次通讯可以在 $\Theta(n/p)$ 时间内完成。总共需要 $\Theta((n\log p)/p)$.时间

- 在第二阶段，每个奇偶步都需要 $\Theta(n/p)$时间。
- 该算法的并行执行时间为

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right)}^{\text{first phase}} + \overbrace{\Theta\left(l\frac{n}{p}\right)}^{\text{second phase}}. \qquad (3)$$

快速排序

快速排序

- 快速排序是串行计算机中最常用的排序算法。原因是它简单，低开销，并且拥有最优的平均时间复杂度。

- 快速排序在序列中选择一个主元，并将序列分为两部分-一部分中所有元素都小于主元，另一部分则大于主元。

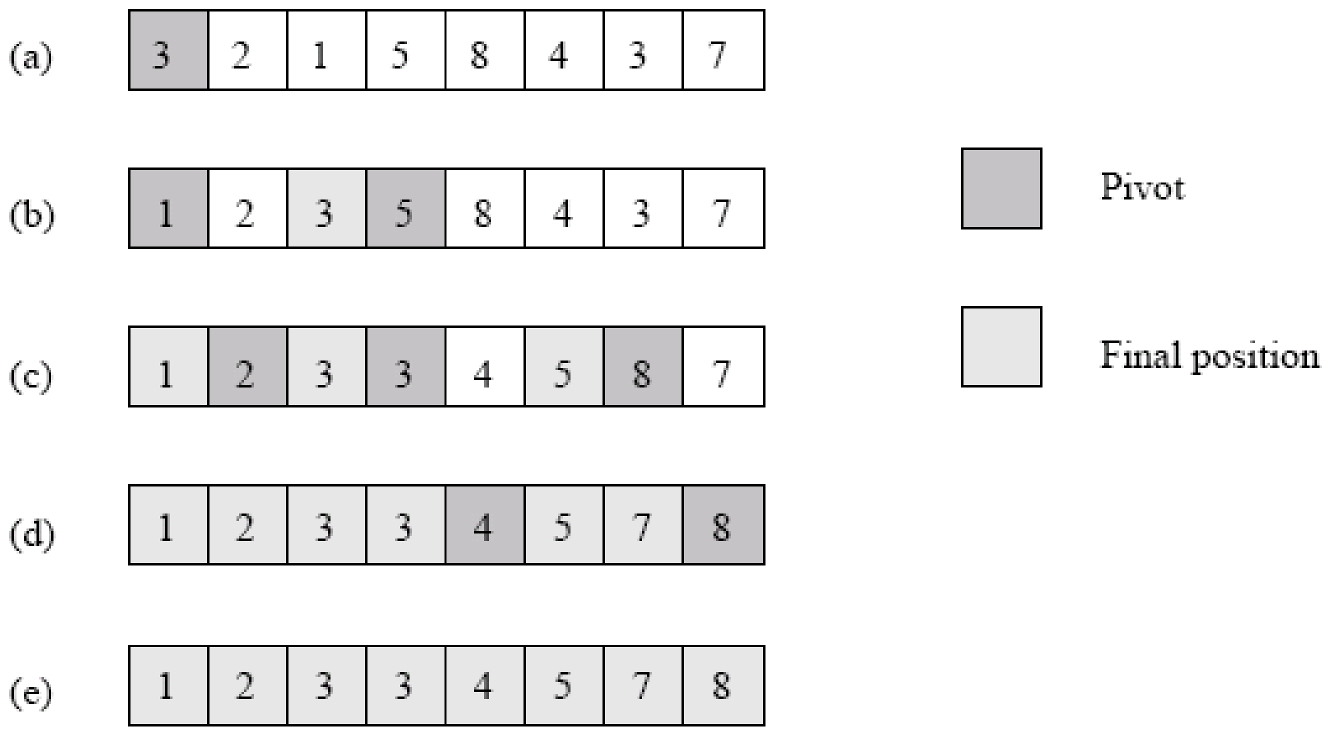
- 这个过程被反复的运用在分开后的两个子序列上。

快速排序

```
procedure QUICKSORT (A, q, r)
begin
    if q < r then
    begin
        x := A[q];
        s := q;
        for i := q + 1 to r do
            if A[i] ≤ x then
            begin
                s := s + 1;
                swap(A[s], A[i]);
            end if
        swap(A[q], A[s]);
        QUICKSORT (A, q, s);
        QUICKSORT (A, s + 1, r);
    end if
end QUICKSORT
```

串行快速排序

快速排序



使用快速排序算法进行 n = 8的序列的排序

快速排序

- 快速排序的性能主要决定于主元的选择。
- 在最好的划分中，主元应该能够划分序列并使最长的序列不超过αn 个元素（对于常数 α）
- 在这个例子中，快速排序的复杂度为$O(n\log n)$.

并行快速排序

- 首先从递归的分解入手：序列被串行地划分，每个子问题由不同的进程来解决。
- 这个算法的下界是 $\Omega(n)$!
- 我们是否可以将划分步骤并行化呢？--实际上，如果能使 n 个进程在 $O(1)$ 时间内依据同一主元划分长为n 的序列，就成功了。
- 在一个真实的机器上，这是非常难做到的。

并行快速排序：PRAM形式

- 考虑CRCW PRAM机器：可以对同一存储单元并发读写的机器
- 这个过程可以视为构造一个进程池。开始时刻，所有进程被放入同一个池子中，每个进程都有一个元素。
- 每个进程都试图将自己的元素写入同一个位置（每个池子都有一个位置）
- 在完成写后，每个进程从该位置读，如果读入的值大于进程持有的元素值，则进程将自己归入“左”池子，否则归入“右”池子。
- 接着每个池子再递归地执行这个操作
- 注意，算法实际生成了一个挂满主元的二叉树。树的深度就是并行执行时间。平均值为 $O(\log n)$.

并行快速排序：PRAM形式

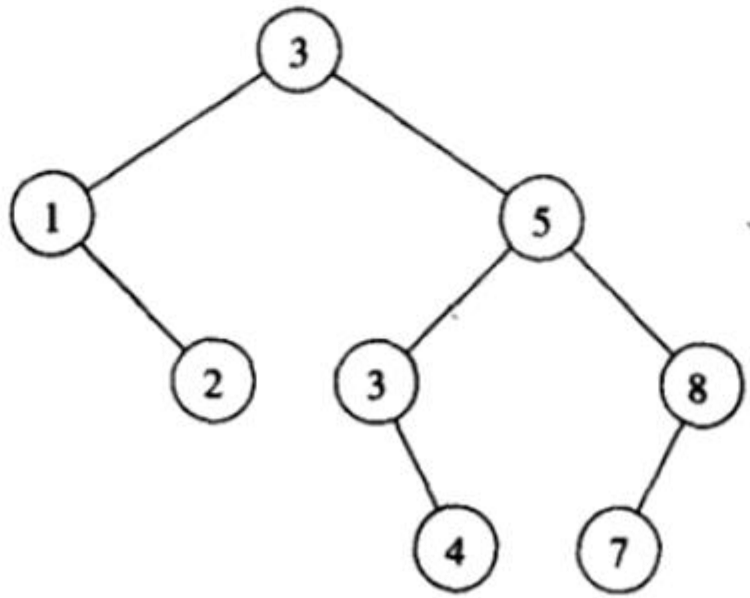


图16 由执行快速排序算法产生的二叉树

注：树的每一层代表一个不同的数组划分迭代。如果主元选择是最优的，那么树的高度是$\Theta(\log n)$，这也是迭代的次数。

并行快速排序：PRAM形式

a)

	1	2	3	4	5	6	7	8
	33	21	13	54	82	33	40	72

b) root = 4

c)

	1	2	3	4	5	6	7	8
leftchild				1				
rightchild				5				

d)

	1	2	3	4	5	6	7	8
leftchild	2			1	8			
rightchild	6			5				

e)

	1	2	3	4	5	6	7	8
leftchild	2	3		1	8			
rightchild	6			5		7		

f)

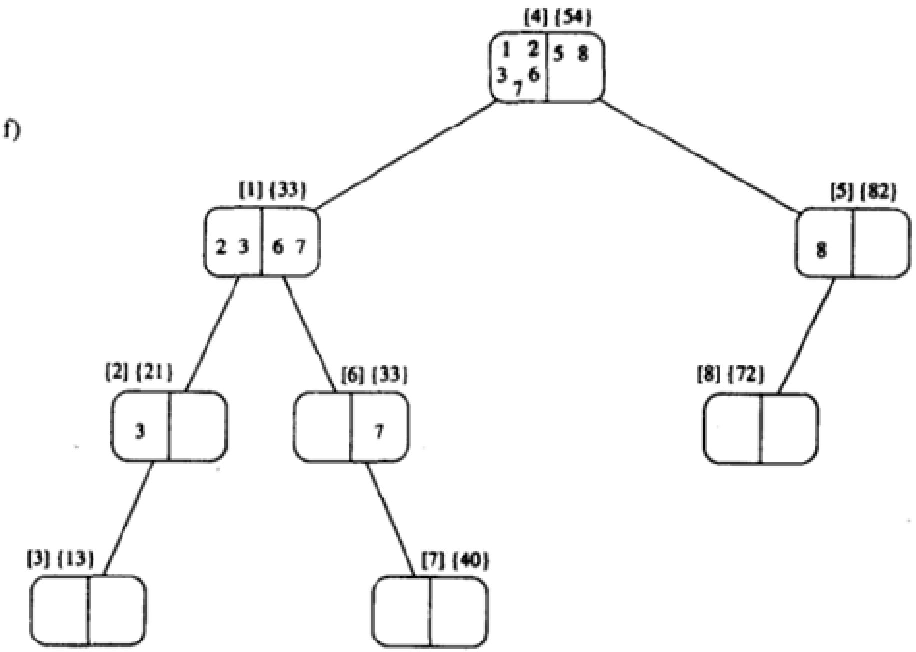
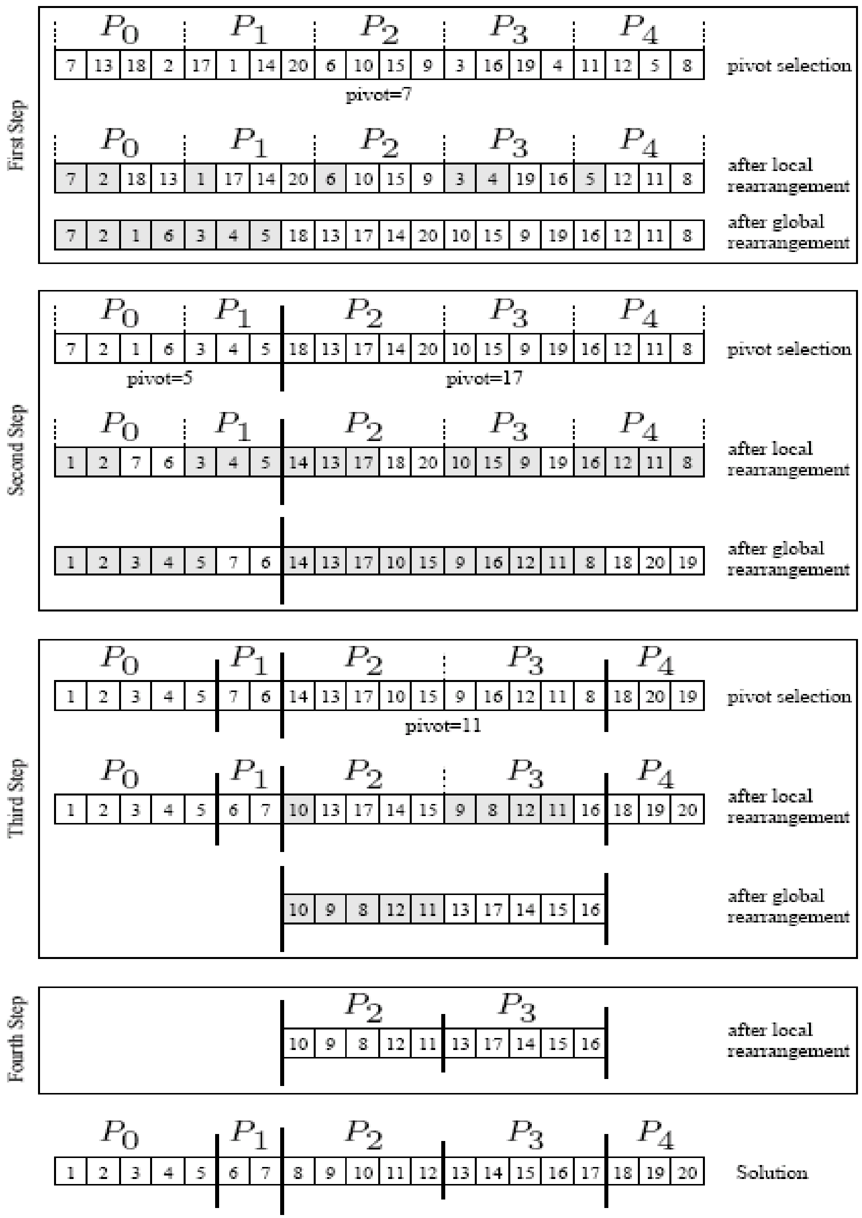


图17 对a)中所示数组执行PRAM算法的情况

并行快速排序：共享地址空间形式

- 考虑一个长度为 n 的序列等分给 p 个处理器
- 一个处理器选择主元并将它广播给其他所有处理器
- 每个处理器按照主元在局部将序列划分为两部分，记为L_i 和 U_i,
- 所有处理器上的L_i 被合并在一起，相应地，U_i, 也被合并在一起
- 接着，处理器被划分为两部分（按照 L 和U 的大小比例划分）来相应的对 L 和U 进行排序
- 将这个过程会被递归地应用在子序列上。

共享地址空间形式


First Step
Second Step
Third Step
Fourth Step
pivot selection
pivot=7
after local rearrangement
after global rearrangement
pivot=5
pivot=17
pivot=11
Solution

并行快速排序：共享地址空间形式

- 我们还没有仔细描述的唯一一件事是：如何将本地的序列通过全局合并得到 *L* 和*U*
- 问题是，如何准确定位局部序列在*L* 或*U* 应该摆放的起始位置

- 每个进程局部地分别计算小于和大于主元的元素个数。
- 使用两次类似于前缀和的操作即可计算出准确的起始位置。
- 在得出准确位置后，就可以进行无冲突地安全写了。

并行快速排序：共享地址空间形式

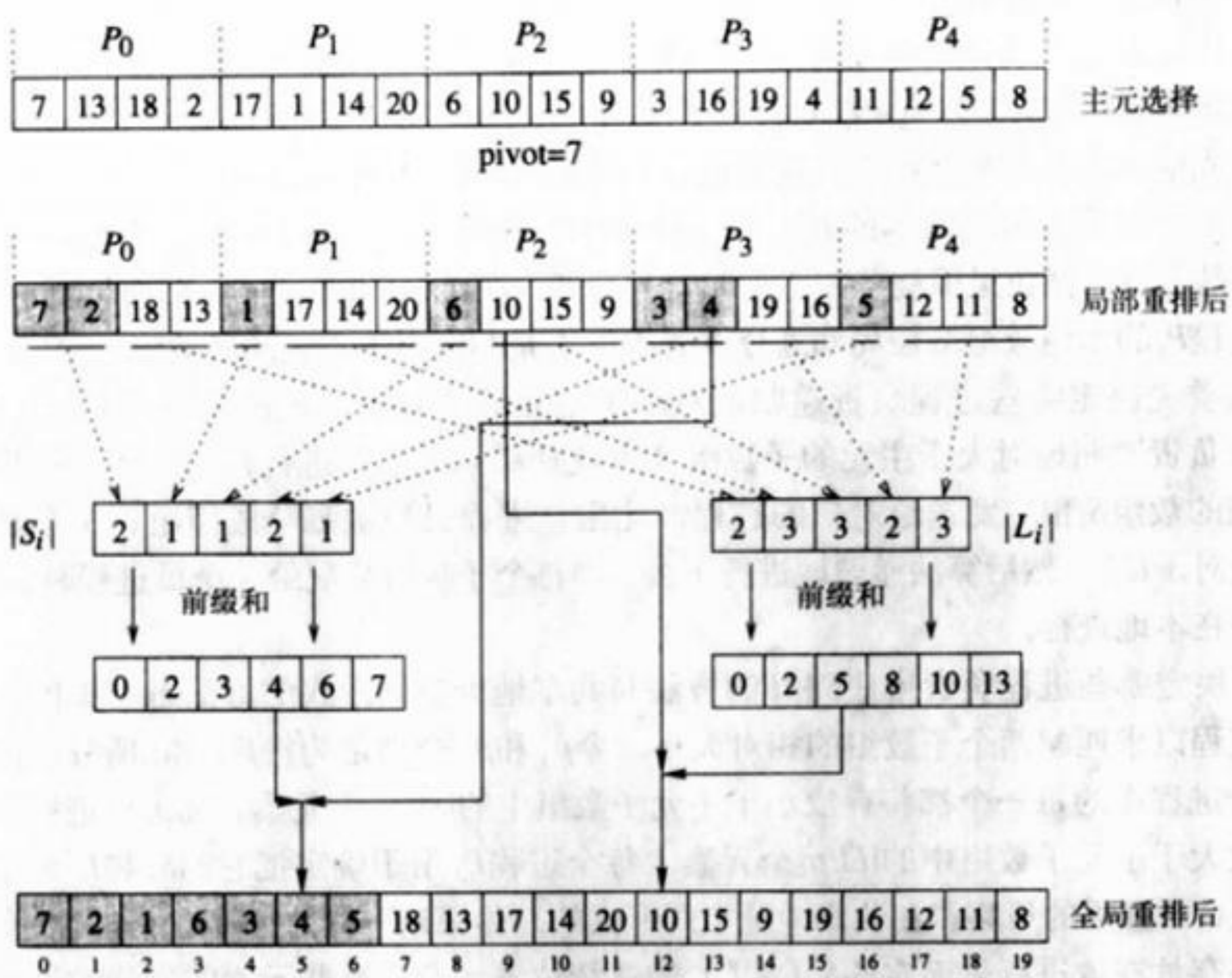


图19 数组的有效全局重排

并行快速排序：共享地址空间形式

- 并行执行时间取决于划分和合并花费的时间，同样地，还有主元的选取。
- 后者与并行化的问题无关，所以我们主要关注前者，假定拥有理想的主元选择方法。

- 首先，这个算法需要不断地执行4步： (i)确定并广播主元，(ii) 依据主元进行局部划分，(iii) 确定本地元素在合并后将要存放的位置； (iv) 进行全局的合并（重排）。
- 第一步耗时 $\Theta(\log p)$,，第二步 $\Theta(n/p)$，第三步$\Theta(\log p)$，以及第四步$\Theta(n/p)$
- 对 n 元素的数组执行分裂工作复杂度为 $\Theta(n/p) + \Theta(\log p)$

并行快速排序：共享地址空间形式

- 执行上述的4步，直到序列被分为 p 份时，问题就退化为局部的串行排序了。
- 因此，总体地并行执行时间为：

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right) + \Theta(\log^2 p)}^{\text{array splits}}. \qquad (4)$$

- 相应的等效率函数是 $\Theta(p\log^2 p)$ ，它主要被广播和前缀和操作的部分控制。

并行快速排序：消息传递模式

- 该算法的一个简单的消息传递形式是，递归地二分机器。
- 假设前半部分的处理器与后半部分的处理器一一配对，
- 由一个指定的处理器选择并广播主元。
- 每个处理器局部将元素划分为两部分，与前文一样，分为 (L_i),和 (U_i) 。
- 前半部分的处理器将 (U_i)发送给后半部分配对的处理器，相应地，后半部分处理器发送(L_i),。
- 很明显地：在完成这些步骤后，所有小于主元的元素都聚集在前半部分机器上，反之亦然。

并行快速排序：消息传递模式

- 上述的过程一直递归执行，直到每个处理器可以局部排序自己的序列。
- 一次全局的重排中，广播主元耗时 $\Theta(\log p)$，局部划分耗时$\Theta(n/p)$，数据传输耗时$\Theta(n/p)$。

- 注意到，这个时间与共享地址空间形式中的是完全一致的。
- 有一点非常重要：对元素的全局重排工作是一个对网络带宽非常敏感的操作。

桶和并行正则采样排序

桶和并行正则采样排序

- 在桶排序中，输入元素的范围[a,b] 被划分为 m 个大小相等的部分，称为桶。
- 每个元素只属于一个桶。
- 如果元素在区间内是均匀分布的，那么每个桶中的数目是大致相等的。
- 将元素放入桶后，对桶内的元素进行局部排序。
- 这个算法的串行时间为 $\Theta(n\log(n/m))$.。

桶和并行正则采样排序

- 并行桶排序相对来说，比较简单。我们选择 $m = p$.。
- 每个处理器负责一个区间，即一个桶。

- 每个处理器为局部的元素确定它们所属的桶，并将它们发给对应的处理器。
- 这些元素使用一个all-to-all操作即可发送到目的处理器。
- 每个处理器局部的对它收到的元素进行排序。

桶和并行正则采样排序

- 上述算法最重要的部分是如何为处理器分配区间。这需要一个合适的划分点选择策略。
- 这个划分点选择策略首先将 n 个元素划分为m 个块，每个块大小为 n/m ，并且使用快速排序对每个块进行排序。
- 接着，从每个有序的块中，均匀地选择$m-1$ 个元素
- 从所有块中选出的共$m(m-1)$ 个元素作为样本，用来决定桶的划分。

- 使用这个策略，最终得到的单个桶的元素不会超过 $2n/m$.

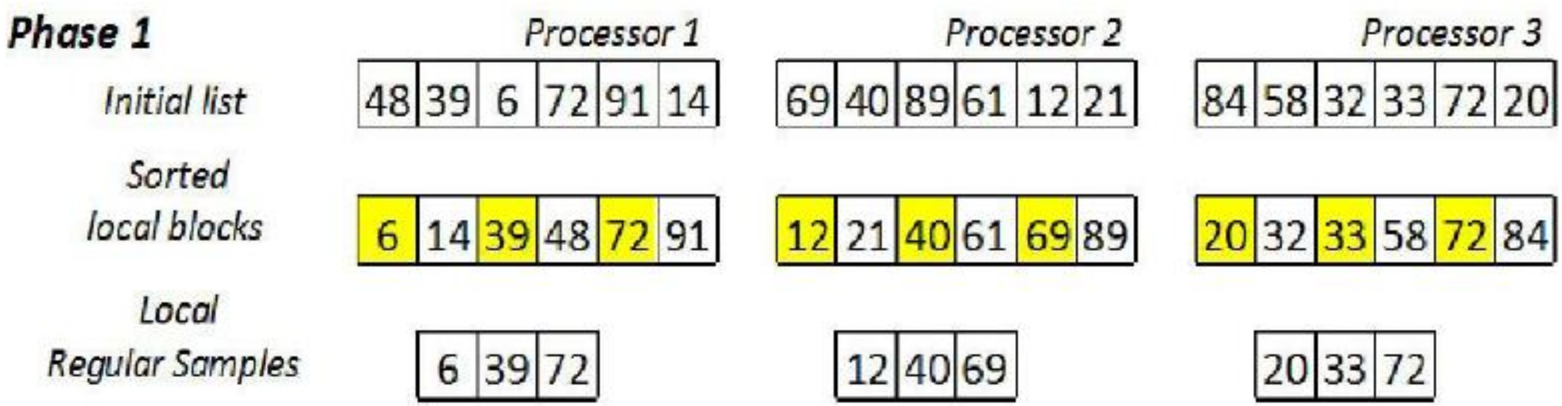
并行正则采样排序

下面详细讲解基于桶排序思想的**并行正则采样排序**

- PSRS(Parallel Sorting by Regular Sampling)
- 一种基于均匀划分的负载均衡的并行排序算法;
- 这种负载均衡是相对的。

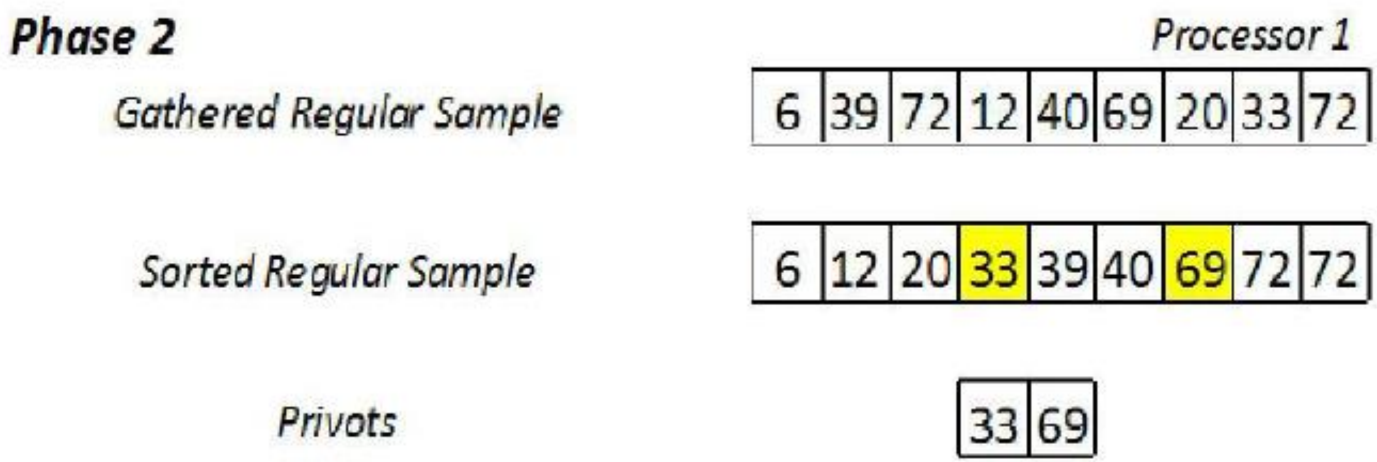
并行正则采样排序

- 第一步：本地数据排序
  - 本地数据排序；
  - 按进程数N等间隔采样。



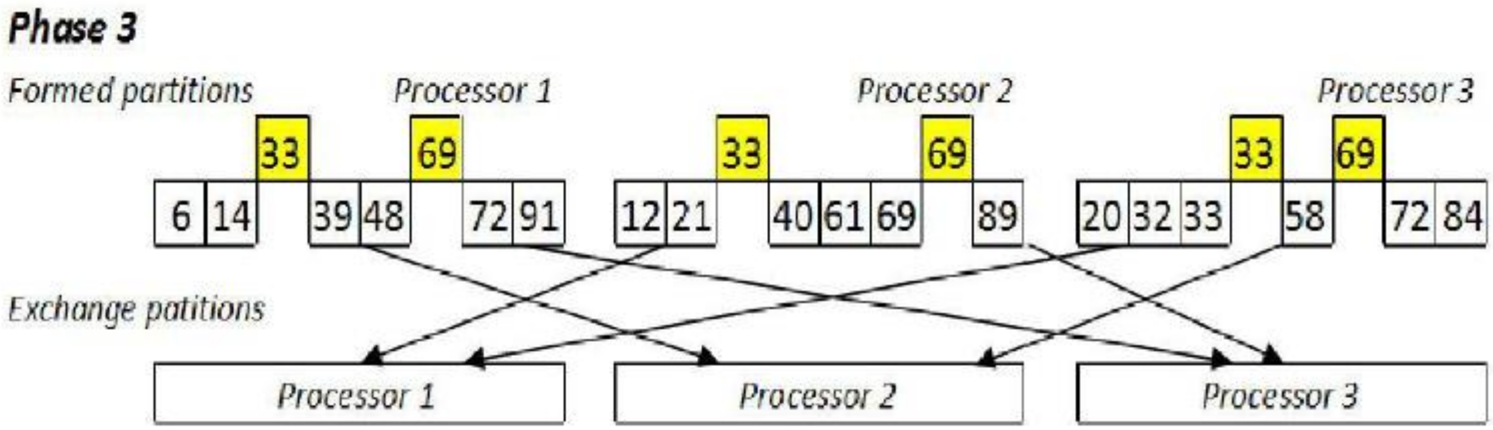
并行正则采样排序

- 第二步：获得划分主元
  - 一个进程收集样本并对所有样本进行排序；
  - 按进程数N对全体样本等间隔采样；
  - 散发最终样本，即主元。



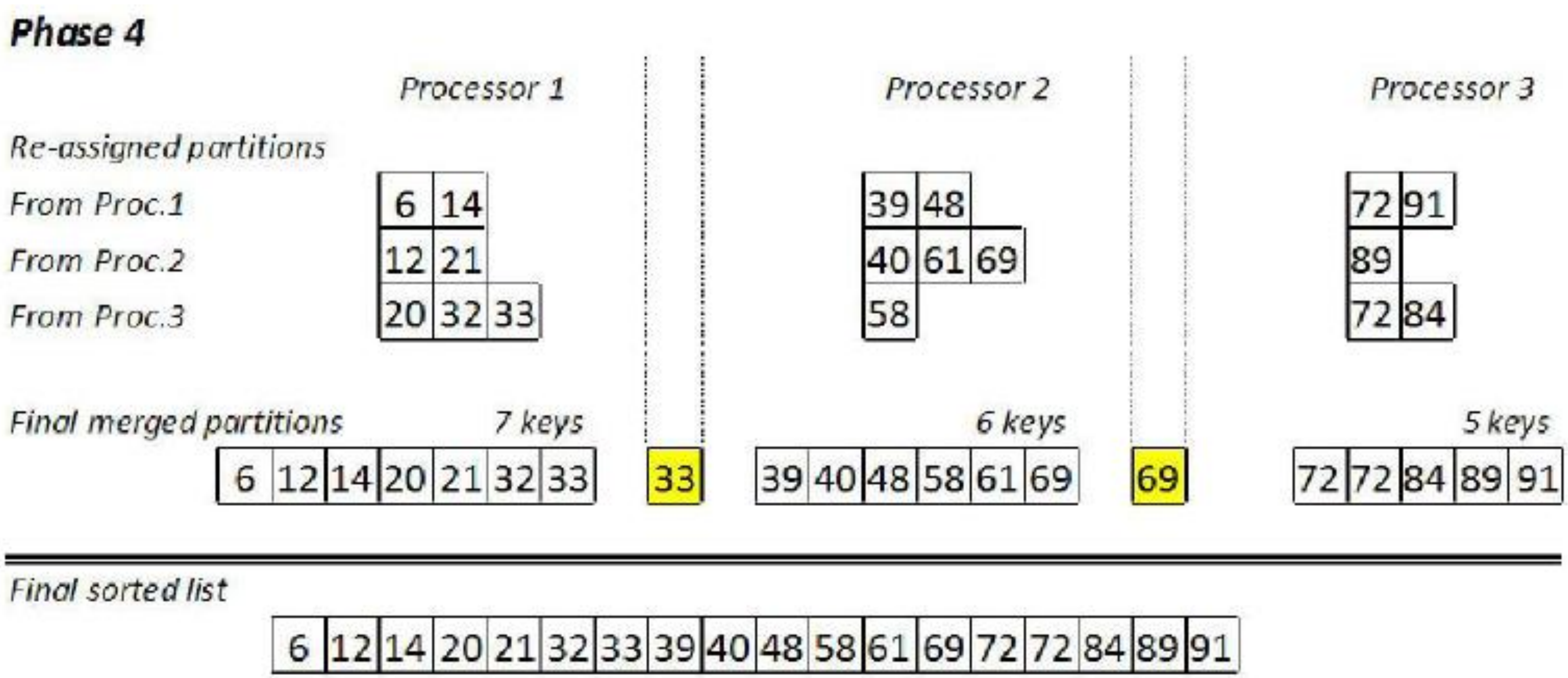
并行正则采样排序

- 第三步：交换数据
  - 交换本地数据划分后的分块；
  - 一个全互换过程。（MPI_Alltoallv）



并行正则采样排序

- 第四步：归并排序
  - 合并得到的分块；
  - 对最终的本地数据进行排序。



并行正则采样排序

- 划分点的选择策略可以被并行化
- 每个处理器局部的生成 $p-1$ 个样本点。
- 所有处理器使用一个 all-to-all操作共享它们的样本点。

- 每个处理器对收到的 $p(p-1)$ 个元素进行排序，并且均匀地选择$p-1$ 个元素作为最终的划分点

并行正则采样排序：分析

- n/p个元素的内部排序耗时$\Theta((n/p)\log(n/p))$，选择 $p-1$ 个样本点耗时 $\Theta(p)$。
- 接着，一个 all-to-all 的广播耗时 $\Theta(p^2)$，对 $p(p-1)$ 个样本点排序耗时$\Theta(p^2\log p)$，选择 $p-1$ 个最终的划分点耗时$\Theta(p)$。
- 每个进程可以将 $p-1$个划分点使用二分查找法插入已经有序的本地序列，这个过程耗时$\Theta(p\log(n/p))$。

- 重排花费的总时间为 $O(n/p)$

并行正则采样排序：分析

- 总时间为：

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(p^2 \log p\right)}^{\text{sort sample}} + \overbrace{\Theta\left(p \log\frac{n}{p}\right)}^{\text{block partition}} + \overbrace{\Theta(n/p)}^{\text{communication}}. \qquad (5)$$

- 该形式的等效率函数为$\Theta(p^3 \log p)$.
- 评价PSRS：
  - 一个好的串行排序算法的时间复杂度为$O(n\log n)$那么，容易证得PSRS算法的时间复杂度在$n>p^3$时为$O(\frac{n}{p}\log n)$；
  - 仍有负载不均匀，在归并排序中排序时间很可能会不均匀。