并行计算
Parallel Computing

主讲 孙经纬

2024年 春季学期

概要

- 第二篇 并行算法的设计
  - 第五章 并行算法与并行计算模型
  - 第六章 并行算法基本设计策略
  - **第七章 并行算法常用设计技术**
  - 第八章 并行算法一般设计过程

第七章 并行算法常用设计技术

- **7.1 划分设计技术**
- 7.2 分治设计技术
- 7.3 平衡树设计技术
- 7.4 倍增设计技术
- 7.5 流水线设计技术

划分设计技术

- 朴素的并行计算思想：
  1. 尽可能将原始问题划分成若干独立、相等的部分
  2. 各部分由相应的处理器同时执行

- 均匀划分技术
- 方根划分技术
- 对数划分技术
- 功能划分技术

均匀划分技术

- 划分方法

  n个元素A[1..n]分成p组，每组A[(i-1)n/p+1..in/p]，i=1~p

- 示例：MIMD-SM模型上的PSRS排序（Parallel Sorting by Regular Sampling)

begin

(1)均匀划分：将n个元素A[1..n]均匀划分成p段，每个p_i处理A[(i-1)n/p+1..in/p]

(2)局部排序：p_i调用串行排序算法对A[(i-1)n/p+1..in/p]排序

(3)选取样本：p_i从其有序子序列A[(i-1)n/p+1..in/p]中选取p个样本元素

(4)样本排序：用一台处理器对p^2个样本元素进行串行排序

(5)选择主元：用一台处理器从排好序的样本序列选取p-1个主元，播送给其他p_i

(6)主元划分：p_i按主元将有序段A[(i-1)n/p+1..in/p]划分成p段

(7)全局交换：各处理器将其有序段按段号交换到对应的处理器中

(8)归并排序：各处理器对接收到的元素进行归并排序

end

均匀划分技术

- 例7.1 PSRS排序过程。N=27， p=3， PSRS排序如下：

(a) 均匀划分：	15	46	48	93	39	6	72	91	14	36	69	40	89	61	97	12	21	54	53	97	84	58	32	27	33	72	20
(b) 局部排序：	6	14	15	39	46	48	72	91	93	12	21	36	40	54	61	69	89	97	20	27	32	33	53	58	72	84	97

(c) 正则采样： 6 39 72 12 40 69 20 33 72

(d) 采样排序： 6 12 20 33 39 40 69 72 72

(e) 选择主元： 33 69

(f) 主元划分：	6	14	15	39	46	48	72	91	93	12	21	36	40	54	61	69	89	97	20	27	32	33	53	58	72	84	97
(g) 全局交换：	6	14	15	12	21	20	27	32	33	39	46	48	36	40	54	61	69	53	58	72	91	93	89	97	72	84	97
(h) 归并排序：	6	12	14	15	20	21	27	32	33	36	39	40	46	48	53	54	58	61	69	72	72	84	89	91	93	97	97

方根划分技术

- 划分方法

  n个元素A[1..n]分成A[(i-1)n^(1/2)+1..in^(1/2)]，i=1~n^(1/2)

- 示例：Valiant归并

将两个长度分别为p, q的有序序列合并为长度p+q的有序序列

通常作为其他算法中的子过程

方根划分技术

- SIMD-CREW模型上的 $k=\lfloor\sqrt{pq}\rfloor$ Valiant归并

//有序组A[1..p]、B[1..q], (假设p<=q), 处理器数 $k=\lfloor\sqrt{pq}\rfloor$

begin

(1)方根划分: A,B分别按 $i\lceil\sqrt{p}\rceil$ 和 $j\lceil\sqrt{q}\rceil$分成若干段（$i=1\sim\lfloor\sqrt{p}\rfloor$、 $j=1\sim\lfloor\sqrt{q}\rfloor$）

(2)段间比较: A划分元与B划分元比较(至多$\lfloor\sqrt{p}\rfloor\cdot\lfloor\sqrt{q}\rfloor$对),

确定A划分元应插入B中的区段;

(3)段内比较: A划分元与B相应段内元素进行比较，并插入适当的位置;

(4)递归归并: B按插入的A划分元重新分段，与A相应段(A除去原划分元)

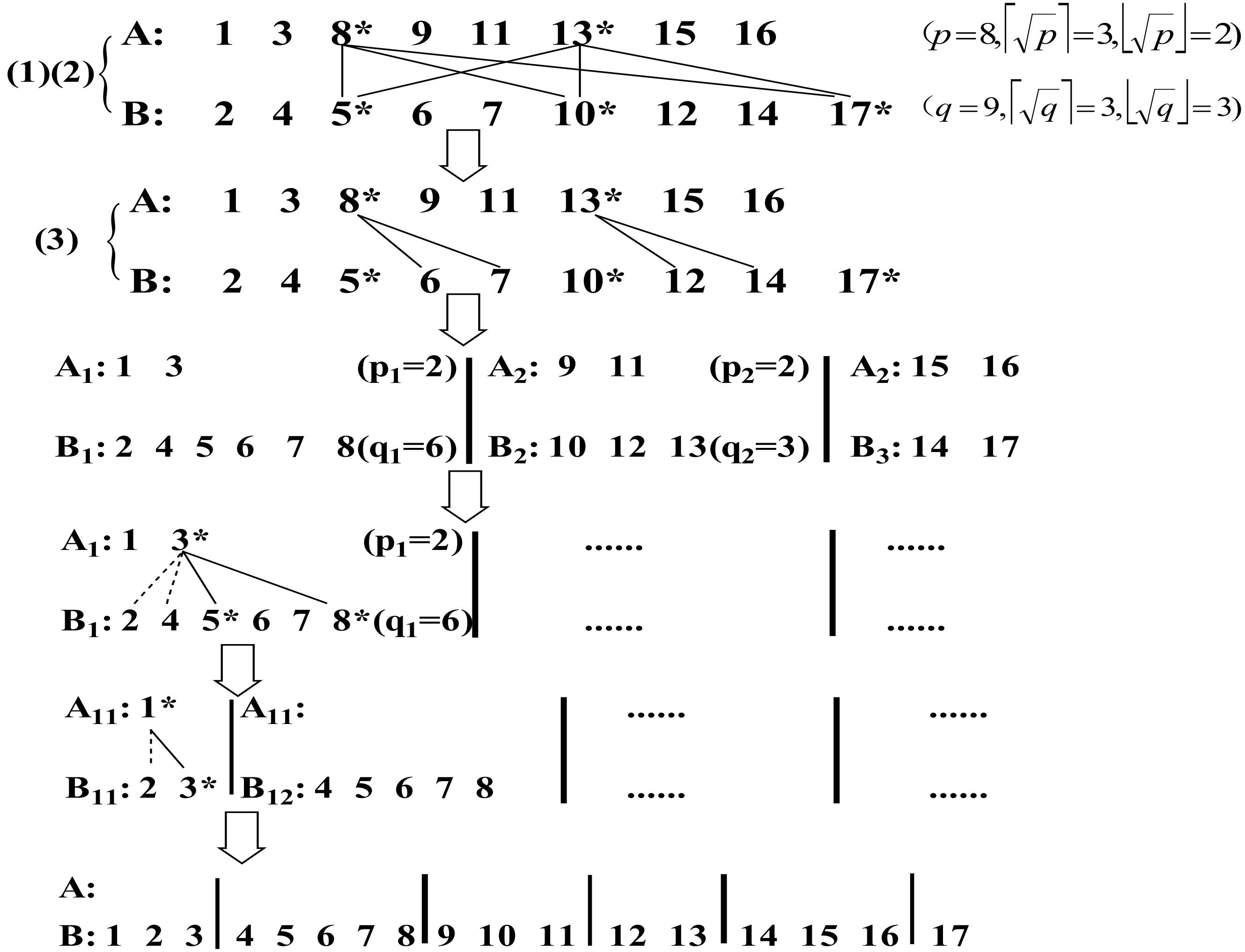
构成了成对的段组，对每对段组递归执行(1)~(3)，直至A

组为0时，递归结束; 各组仍按 $k=\lfloor\sqrt{pq}\rfloor$分配处理器;

end

方根划分技术

- 示例: A={1,3,8,9,11,13,15,16},p=8; B={2,4,5,6,7,10,12,14,17},q=9



$$(x_1^2+x_2^2+\cdots+x_n^2)(y_1^2+y_2^2+\cdots+y_n^2)\ge(x_1y_1+x_2y_2+\cdots+x_ny_n)^2$$

柯西不等式

方根划分技术

- **算法分析**

(1)算法在并行递归过程中所需的处理器数⩽ $k=\lfloor\sqrt{pq}\rfloor$

段间比较: $\lfloor\sqrt{p}\rfloor\cdot\lfloor\sqrt{q}\rfloor$ **比较对数⩽** $\lfloor\sqrt{pq}\rfloor=k$ **;**

段内比较: $\lfloor\sqrt{p}\rfloor\cdot(\lfloor\sqrt{q}\rfloor-1)\le\lfloor\sqrt{pq}\rfloor=k$

递归调用：设 A,B 分成若干子段对为(p_1,q_1), (p_2,q_2),……

则Σp_i⩽p，Σq_i⩽q，　由 Cauchy 不等式=>

$$\sum\lfloor\sqrt{p_iq_i}\rfloor\le\lfloor\sum\sqrt{p_iq_i}\rfloor\le\lfloor\sqrt{\sum p_i\sum q_i}\rfloor\le\lfloor\sqrt{pq}\rfloor=k$$

虽然每个子任务处理器个数不一样，通过这种方根划分，巧妙地将总处理器数控制在k以内

综上，整个过程可用处理器数 $k=\lfloor\sqrt{pq}\rfloor$ **完成。**

(2)时间分析

记λ$_i$是第 i 次递归后的 A 组最大长度，=> $\lambda_0=p$**，** $\lambda_i\le\lfloor\sqrt{\lambda_{i-1}}\rfloor\le\cdots\le\lfloor p^{2^{-i}}\rfloor$

算法在 $\lambda_i=\text{常数}C$ **时终止递归，即** $\text{常数}C\le p^{2^{-i}}\le\text{常数}C+1$ **=>** $i\le\log\log p+\text{常数}C_1$

由(1)知算法中其他各步的时间为 O(1)，所以 Valiant 归并算法时间

$$t_k(p,q)=O(\log\log p)\qquad p\le q$$

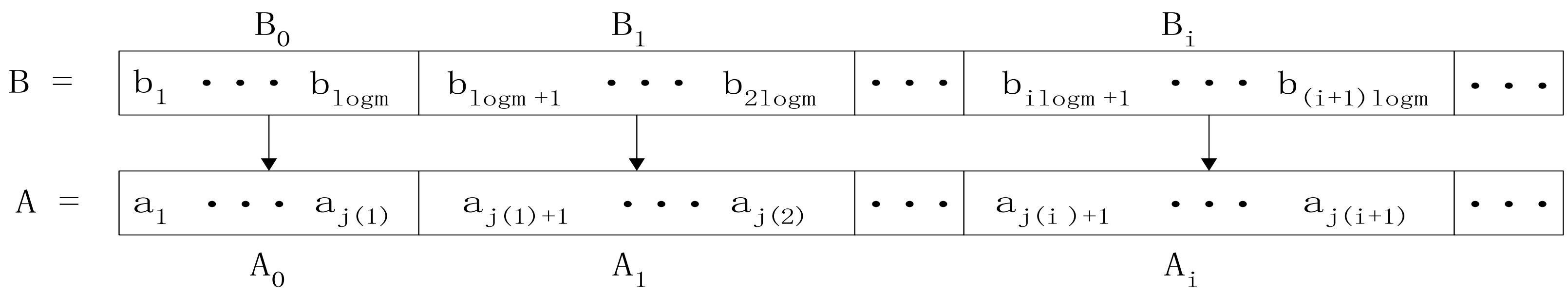
对数划分技术

- 划分方法

  n个元素A[1..n]分成A[(i-1)logn+1..ilogn]，i=1~n/logn

- 示例：PRAM-CREW上的对数划分并行归并排序

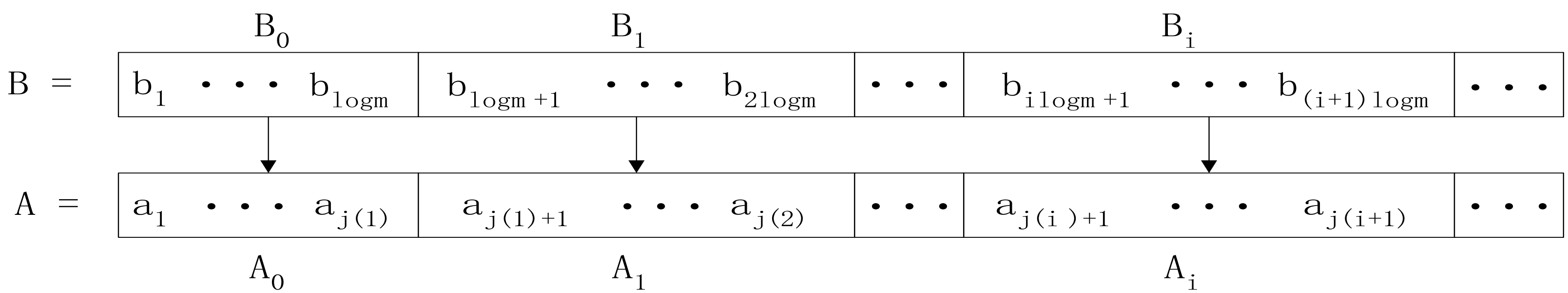
设有序组A[1..n]和B[1..m]



如果在选取A序列中的划分元素时考虑其在B序列中的全局位序，就不必对划分元素实行段间的比较

求一个元素在一个有序序列中的位置，可以使用二分查找

对数划分技术



令$j[i]=rank(b_{ilogm}:A)$为b_{ilogm}在A中的位序，即A中小于等于b_{ilogm}的元素数

例：A=(4,6,7,10,12,15,18,20), B=(3,9,16,21) n=8, m=4

=>logm=log4=2

=> $j[1]=rank(b_{logm}:A)=rank(b_2:A)=rank(9:A)=3$, $j[2]=\ldots=8$

B_0: 3，9　　B_1: 16，21

A_0: 4，6，7　　A_1: 10，12，15，18，20

A和B归并化为(A_0，B_0)和(A_1，B_1)的归并

功能划分技术（???)

- 划分方法

  n个元素A[1..n]分成等长的p组，每组满足某种特性。

- 示例：(m, n)选择问题(求出n个元素中前m个最小者)
  - 功能划分：要求每组元素个数必须大于等于m;
  - 算法：p194算法7.4 输入：A=(a_1,…,a_n); 输出：前m个最小者;

  Begin

  (1) 功能划分：将A划分成g=n/m组，每组含m个元素;

  (2) 局部排序：使用双调排序网络将各组并行进行排序;

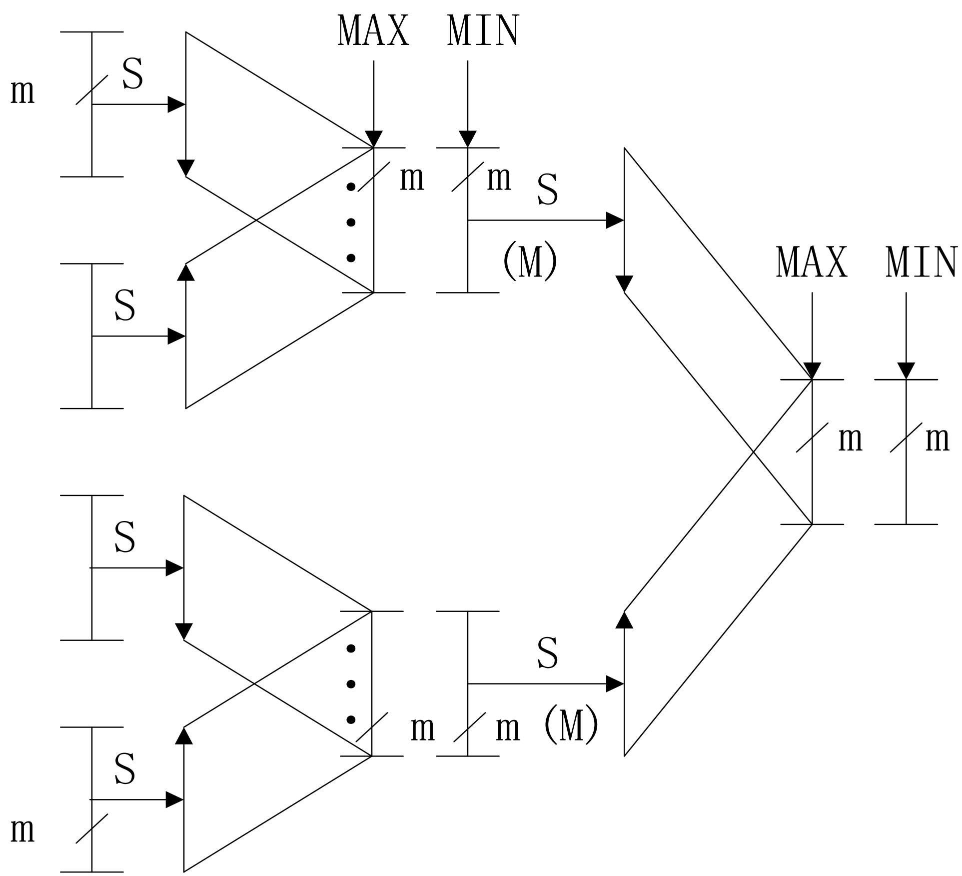
  (3) 构造双调：每两个有序子序列合并成一个双调序列;

  (4) 两两比较：根据Batcher定理，将双调序列划分为MIN, MAX

  (5) 排序-比较：对各个MIN序列，重复执行第(2)-(4)步，直至选出m个最小者。

  End

功能划分技术



(m, n)选择过程，S表示排序，M表示归并

第七章 并行算法常用设计技术

- 7.1 划分设计技术
- **7.2 分治设计技术**
- 7.3 平衡树设计技术
- 7.4 倍增设计技术
- 7.5 流水线设计技术

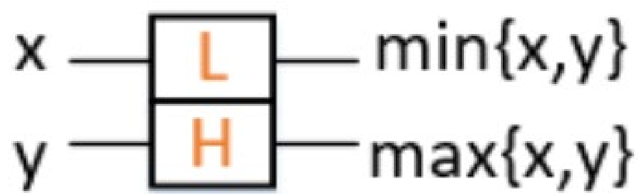
并行分治设计步骤

- 将输入划分成若干个规模相等的子问题;
- 同时(并行地)递归求解这些子问题;
- 并行地归并子问题的解，直至得到原问题的解。

双调排序网络

- Batcher比较器

在两个输入端给定输入x,y，再在两个输出端输出max{x,y}和min{x,y}



双调排序网络

- 双调序列（Bitonic sequence）

一个序列$x_0,x_1,\ldots,x_{n-1}$是双调序列，如果：

(1)存在一个x_k $(0 \le k \le n-1)$, 使得$x_0 \ge \ldots \ge x_k \le \ldots \le x_{n-1}$成立；或者

(2)序列能够循环移位满足条件(1)

(1,3,5,7,8,6,4,2,0)

(8,7,6,4,2,0,1,3,5)

(1,2,3,4,5,6,7,8)

以上都是双调序列

“山谷”和“山峰”都是双调序列

双调排序网络

- Batcher定理

给定双调序列$(x_0,x_1,\ldots,x_{2n-1})$，令$s_i=\min\{x_i, x_{i+n}\}$和 $l_i=\max\{x_i,x_{i+n}\}$，则小数序列$(s_0,s_1,\ldots,s_{n-1})$和大数序列$(l_0,l_1,\ldots,l_{n-1})$满足：

(1) $s_i \le l_j \quad (0\le i, j\le n-1)$

(2)小数和大数序列仍是双调序列

- 直观理解：

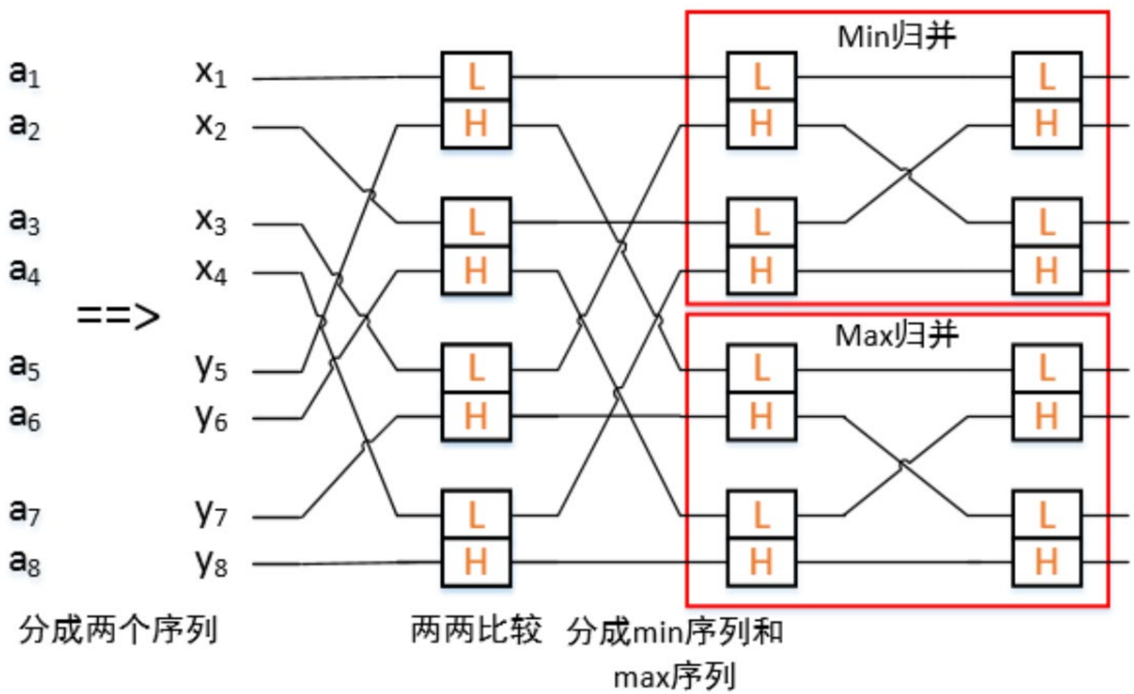
由于可以循环移位，所以“中间”其实可以是任意位置

将任意一个长为2n的双调序列A从中间切成两半，分成等长的两个序列s和l，然后s和l相同位置的元素s_i与l_i比较，小的放到Min序列，大的放到Max序列。由此得到的Max序列和Min序列也是双调序列，且Min序列的每个元素小于或等于Max序列的每个元素。

双调排序网络

- 基于Batcher定理的双调归并网络

　　将任意一个长为2n的双调序列划分成Min序列和Max序列，再分别对Min序列和Max序列进行划分，以此类推，直到n=1。最后再将所有含两个元素的子序列归并成完全有序的序列



双调排序网络

- 双调归并网络

输入：双调序列$X=(x_0,x_1,\ldots,x_{n-1})$

输出：非降有序序列$Y=(y_0,y_1,\ldots,y_{n-1})$

注意！双调归并网络将一个**双调序列**转换为有序序列

Procedure BITONIC_MERG(x)

Begin

(1)for i=0 to n/2-1 par-do

(1.1) $s_i=\min\{x_i,x_{i+n/2}\}$

(1.2) $l_i=\max\{x_i,x_{i+n/2}\}$

end for

(2)Recursive Call:

(2.1)BITONIC_MERG(MIN=$(s_0,\ldots,s_{n/2-1})$)

(2.2)BITONIC_MERG(MAX=$(l_0,\ldots, l_{n/2-1})$)

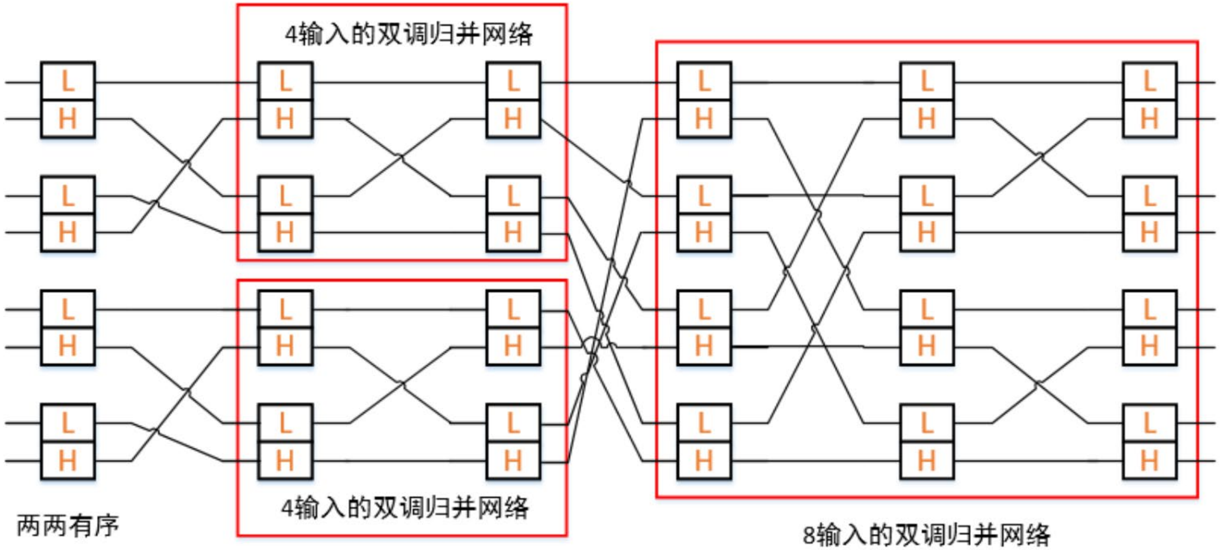
(3)output sequence MIN followed by sequence MAX

End

双调排序网络

双调排序网络通过调用双调归并网络，实现对**任意序列**的并行排序

- 双调排序网络 —— 给定2n长度任意序列：
  1. 两两比较，形成n个2元素有序序列
  2. 每两个2元素有序序列，可构造成一个4元素双调序列
  3. 使用双调归并网络，将4元素双调序列变为有序序列
  4. 每两个4元素有序序列，可构造成一个8元素双调序列
  5. 使用双调归并网络，将8元素双调序列变为有序序列
  6. 按步骤2-5类推，直到将2n个元素变为有序序列



第七章 并行算法常用设计技术

- 7.1 划分设计技术
- 7.2 分治设计技术
- **7.3 平衡树设计技术**
- 7.4 倍增设计技术
- 7.5 流水线设计技术

平衡树设计步骤

- 设计思想

　　以树的叶结点为输入，中间结点为处理结点，由叶向根或由根向叶逐层进行并行处理。

- 示例
  - 求最大值
  - 计算前缀和

求最大值

- 算法7.8：SIMD-TC(SM)上求最大值算法 TC：树形连接网络

Begin

for k=m-1 to 0 do

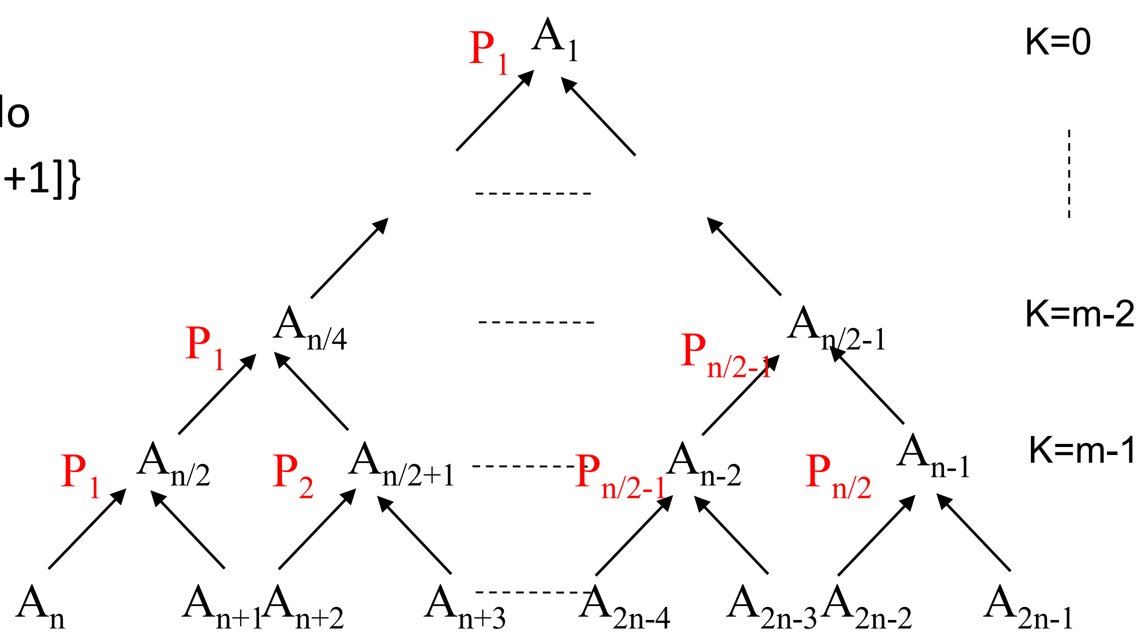
for j=2^k to 2^{k+1}-1 par-do

A[j]=max{A[2j], A[2j+1]}

end for

end for

end



- 时间分析

$t(n)=m\times O(1)=O(\log n)$

$p(n)=n/2$

求最大值

CRCW模型上常常能构造出这种“抽象的”常数时间算法，然而真实机器其实很难满足CRCW

- SIMD-CRCW上常数时间求最大值算法

算法 SIMD-CRCW上枚举算法

//输入A[1..p]， p个不同元素

//B[1..p][1..p],M[1..p]为中间处理用的布尔数组, 如果M[i] = 1, 则A[i]为最大值

```
begin
   (1)for 1≤i, j≤p par-do   //工作量O(p²); 时间O(1),因为允许同时读
          if A[i]≥A[j] then B[i, j]=1 else B[i, j]=0
          end if
      end for
   (2)for 1≤i≤p par-do      //工作量O(p²); 时间O(1),因为允许同时写
          M[i]=B[i,1]∧ B[i,2]∧… ∧B[i,p]
      end for
end
```

```
M[i]=1
for 1≤j≤p par-do
   if B[i, j]==0 then M[i]=0
```

- **$T(n)=O(1)$**
- **$W(n)=O(p^2)$**
- **可以用p^2个处理器实现**
- **速度虽快，但不是W最优**

计算前缀和

- 问题定义

  n个元素$\{x_1,x_2,\ldots,x_n\}$，前缀和是n个部分和：

  $S_i=x_1*x_2*\ldots*x_i$，$1\le i\le n$　这里*可以是＋或×

- 串行算法：　$S_i=S_{i-1}*x_i$　计算时间为 O(n)
- 并行算法：p199算法7.9 SIMD-TC上非递归算法

令$A[i]=x_i$, i=1~n,

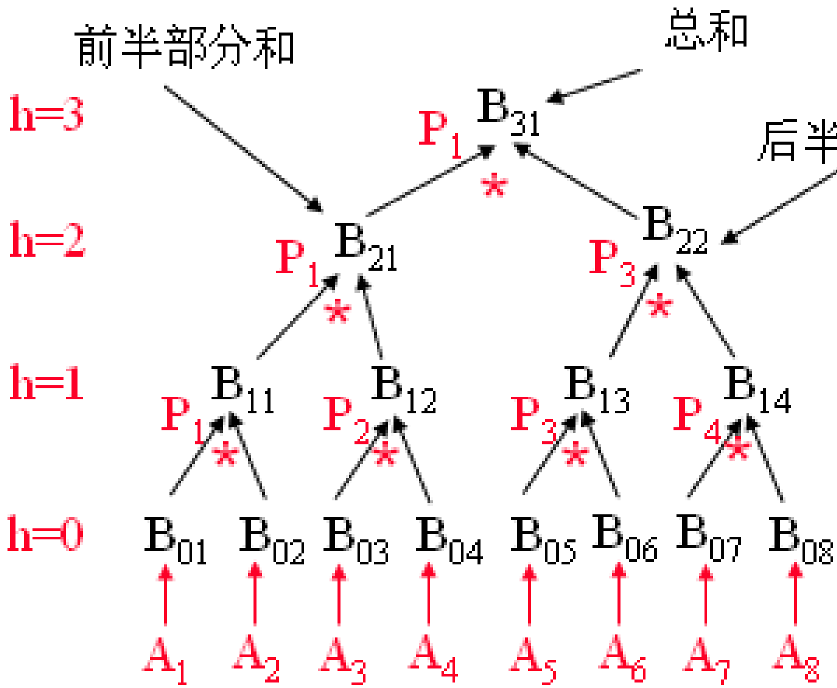
B[h,j]和C[h,j]为辅助数组(h=0~logn, j=1~$n/2^h$)

数组B记录由叶到根正向遍历树中各结点的信息(求和)

数组C记录由根到叶反向遍历树中结点的信息(播送前缀和)

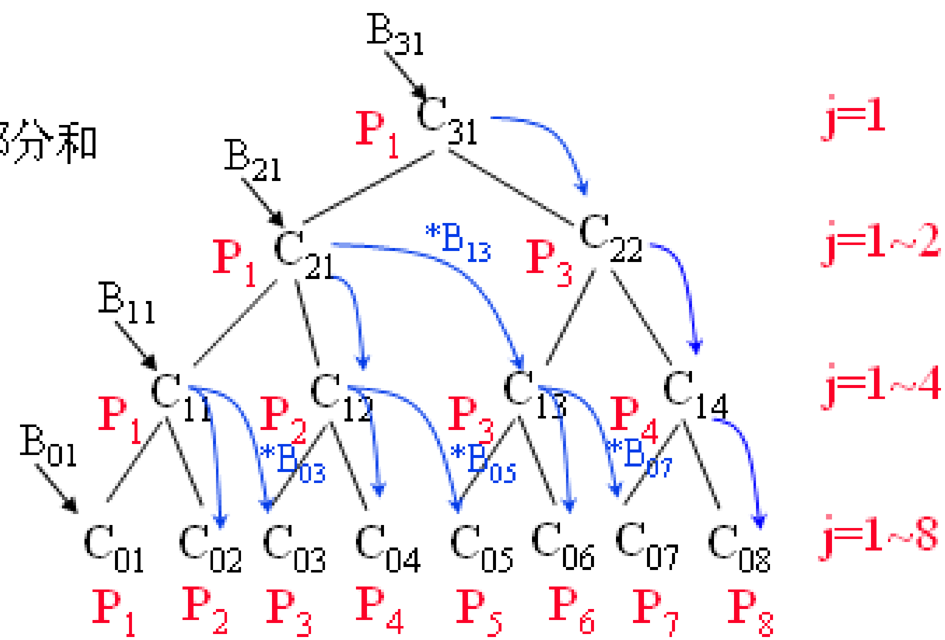
计算前缀和

- 例： n=8, p=8, C01~C08为前缀和



计算：B[h,j]=B[h-1,2j-1]*B[h-1,2j]

正向遍历(求和)



计算：C[h,1]=B[h,1] j=1

C[h,j]=C[h+1, j/2] j=even

C[h,j]=C[h+1,(j-1)/2]*B[h, j] j=odd>1

反向遍历(播送前缀和)

计算前缀和

- SIMD-SM上非递归算法

```
begin
  (1)for j=1 to n par-do    //初始化
        B[0,j]=A[j]
     end if
  (2)for h=1 to logn do   //正向遍历
        for j=1 to n/2h par-do
           B[h,j]=B[h-1,2j-1]*B[h-1,2j]
        end for
     end for
```

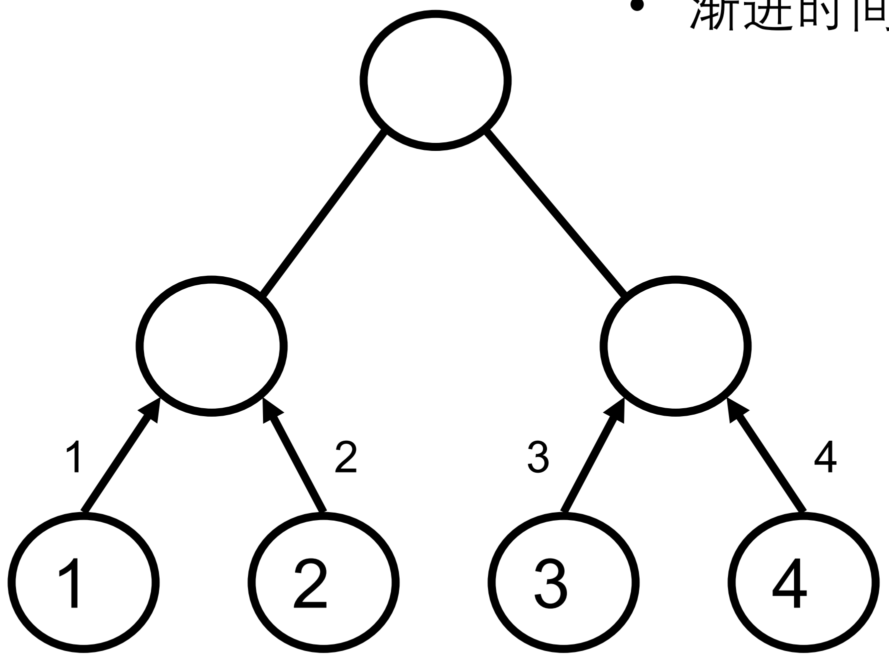
```
(3)for h=logn to 0 do   //反向遍历
      for j=1 to n/2^h par-do
        (i)  if j=even then   //该结点为其父结点的右儿子
                C[h,j]=C[h+1,j/2]
             end if
        (ii) if j=1 then       //该结点为最左结点
                C[h,1]=B[h,1]
             end if
        (iii) if j=odd>1 then   //该结点为其父结点的左儿子
                C[h,j]=C[h+1,(j-1)/2]*B[h,j]
             end if
      end for
   end for
end
```

时间分析:

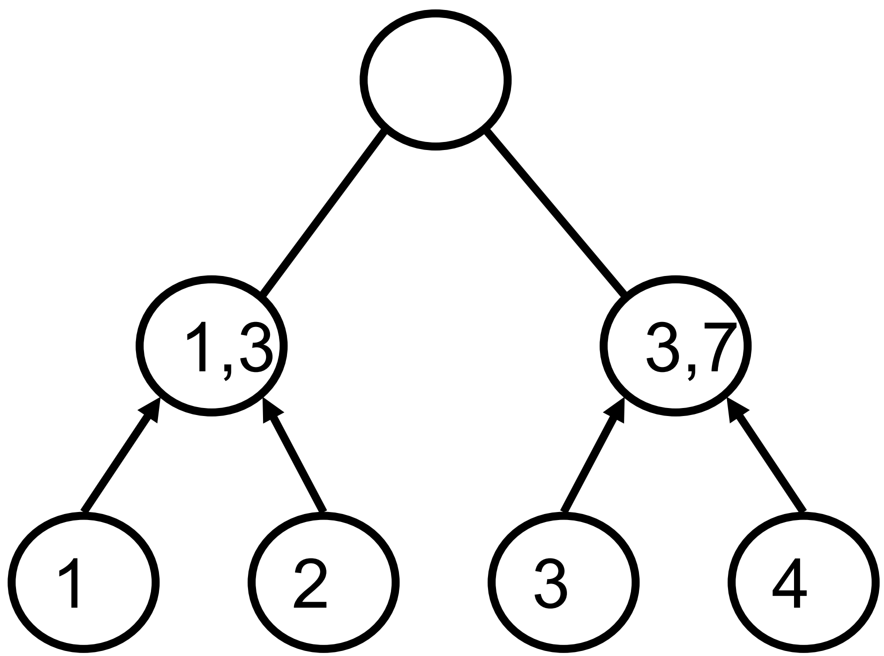
(1) O(1)   (2) O(logn)   (3) O(logn)
==> t(n)=O(logn)，p(n)=n， c(n)=O(nlogn)

计算前缀和

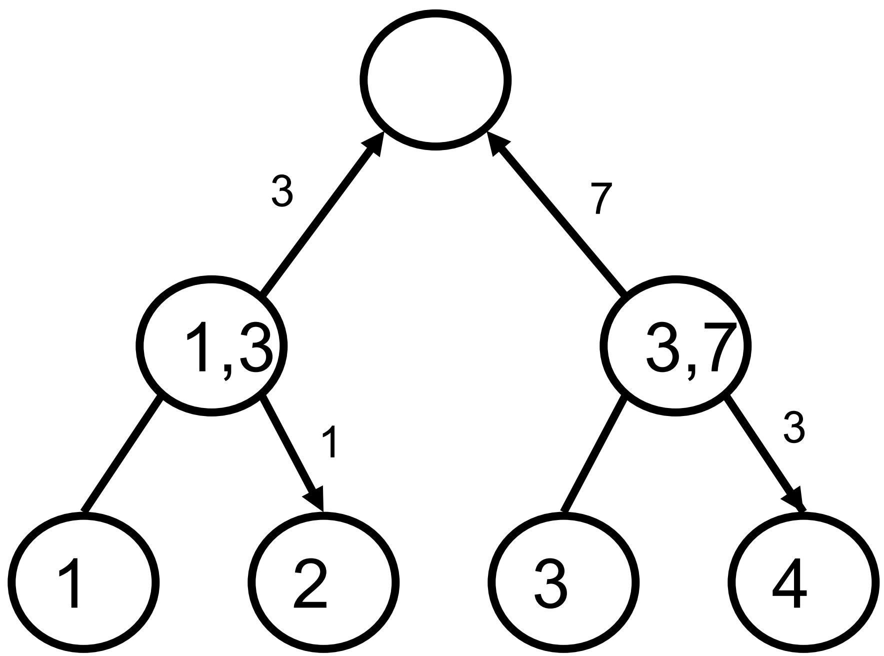
另一种实现方式:
- 不必等正向遍历完成
- 部分结果可以提前开始反向遍历
- 渐进时间复杂度一样



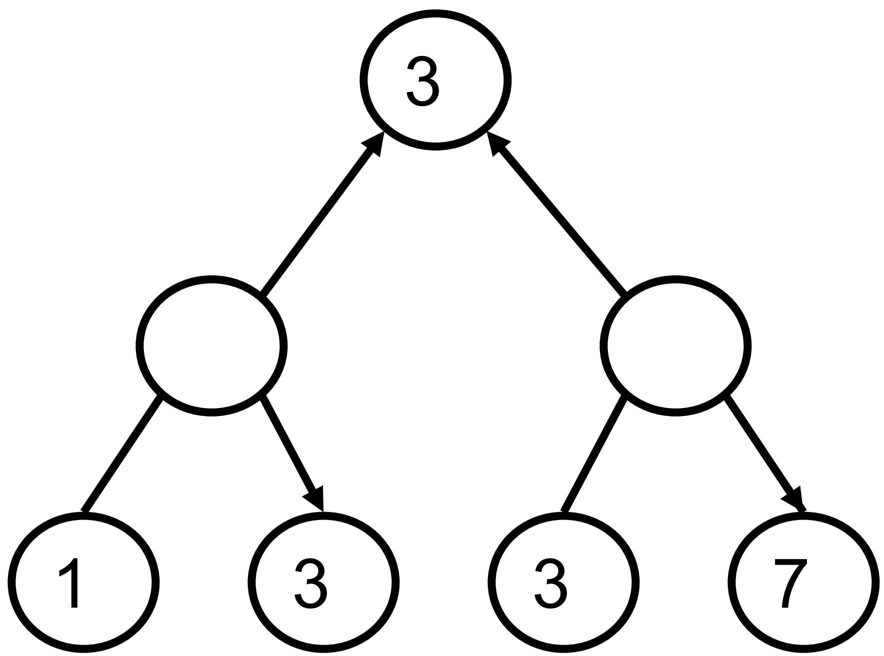
计算前缀和



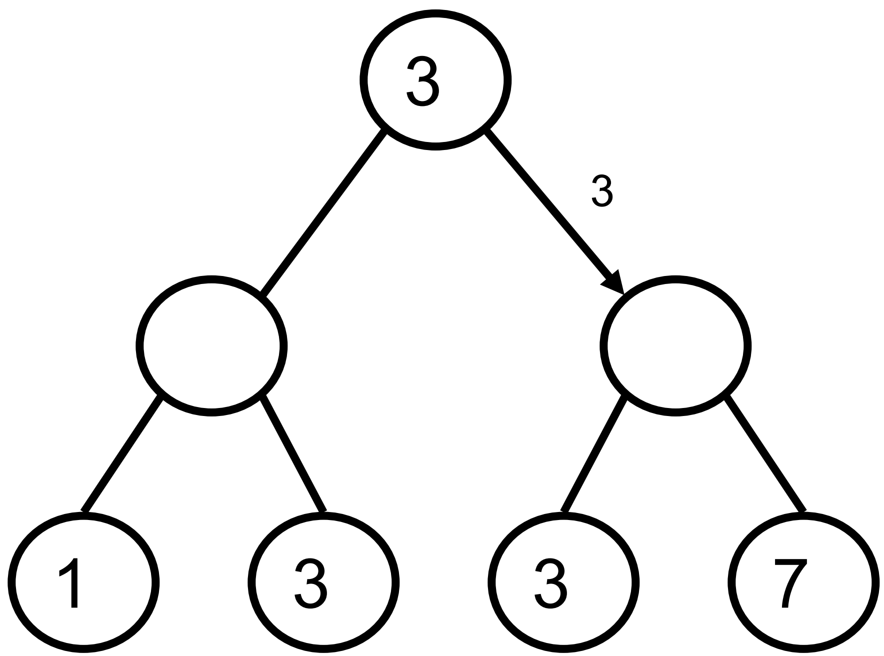
计算前缀和



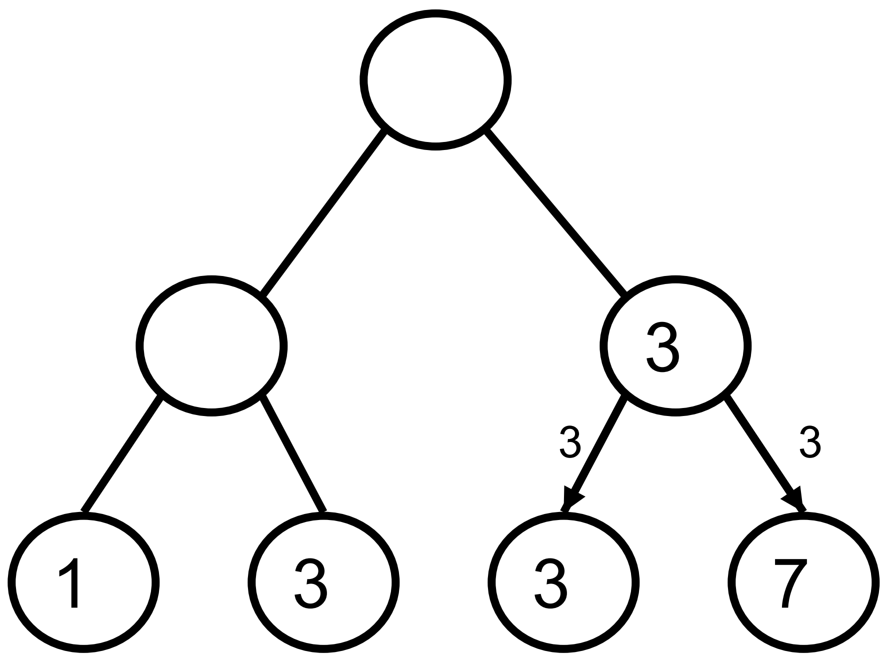
计算前缀和



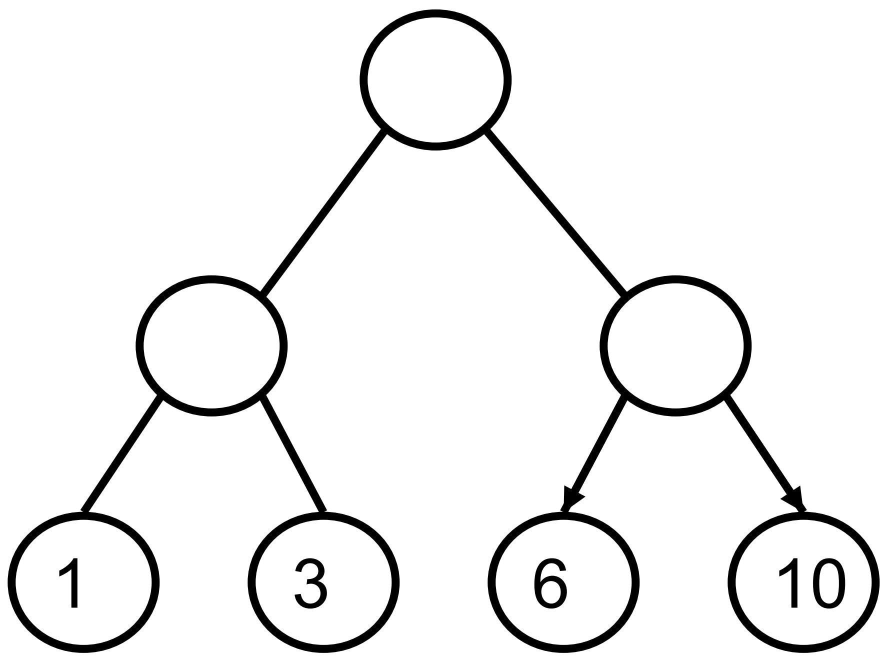
计算前缀和



计算前缀和



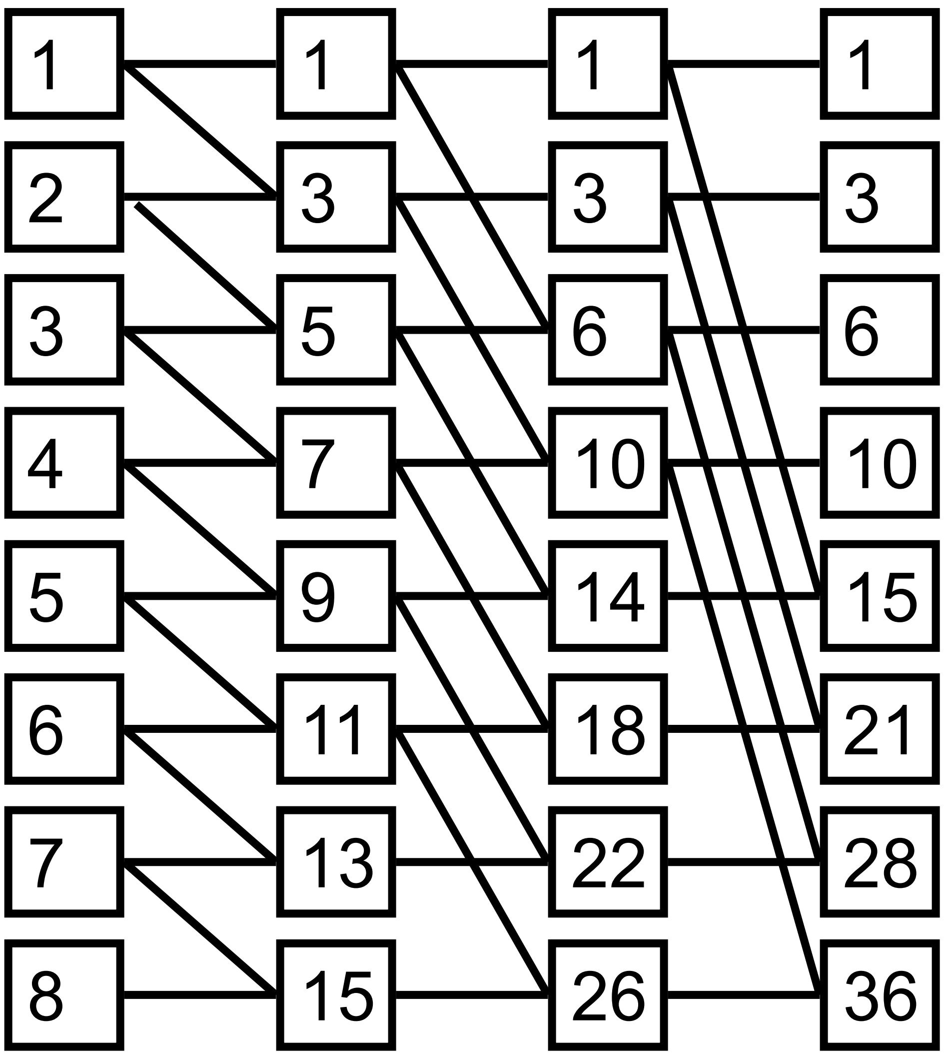
计算前缀和



计算前缀和

- Properties:
  - time: $2 \log n$
  - processors: $2n - 1$
  - cost $O(n \log n)$
- Comparison with PRAM algorithm
  - asymptotically equivalent
  - in practice, less efficient
    - weaker model

计算前缀和



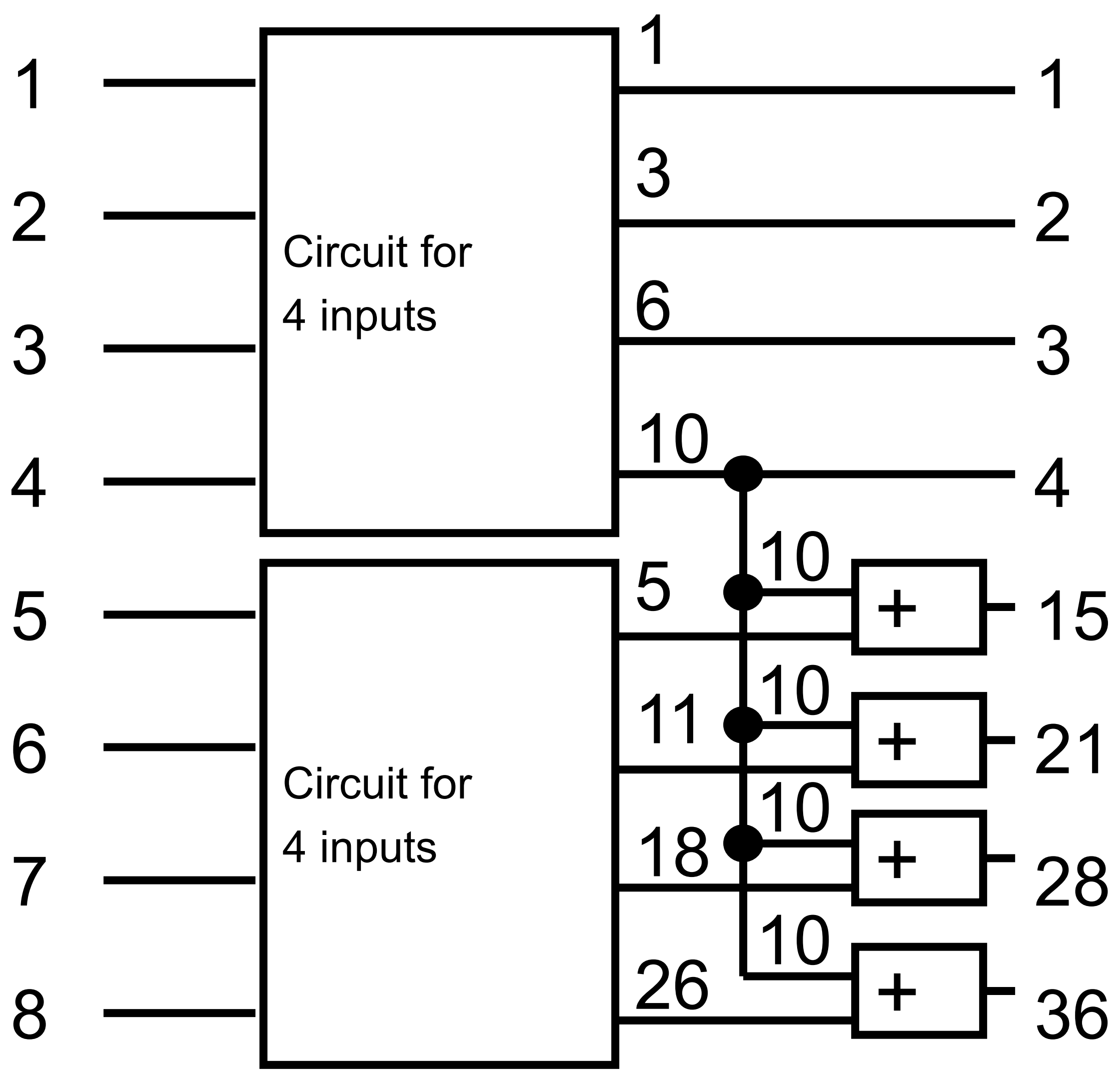
depth (time) = 3

complexity (cost) $= 3 \times 8$

$= 24$

Specialized Circuit

计算前缀和



Recursive Circuit

第七章 并行算法常用设计技术

- 7.1 划分设计技术
- 7.2 分治设计技术
- 7.3 平衡树设计技术
- **7.4 倍增设计技术**
- 7.5 流水线设计技术

倍增设计技术

- 设计思想
  - 又称指针跳跃(pointer jumping)技术，特别适合于处理链表或有向树之类的数据结构;
  - 当递归调用时，所要处理数据之间的距离逐步加倍，经过k步后即可完成距离为2^k的所有数据的计算。
- 示例
  - 表序问题
  - 求森林的根

表序问题

- 问题描述

  n个元素的列表L，求出每个元素在L中的位序rank(k)，rank(k)可视为元素k至表尾的距离；

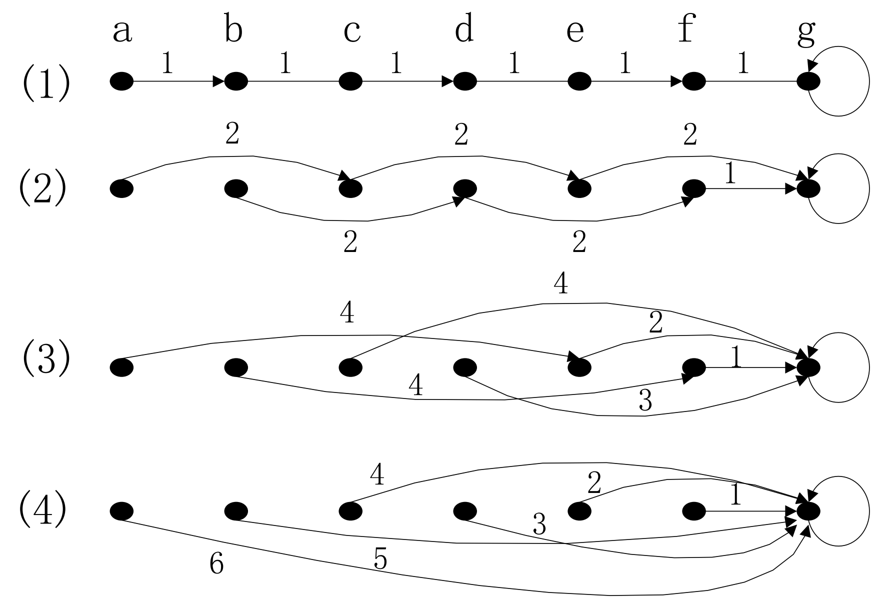
- 示例： n=7

  (1)p[a]=b, p[b]=c, p[c]=d, p[d]=e, p[e]=f, p[f]=g, p[g]=g
  r[a]=r[b]=r[c]=r[d]=r[e]=r[f]=1, r[g]=0

  (2)p[a]=c, p[b]=d, p[c]=e, p[d]=f, p[e]=p[f]=p[g]=g
  r[a]=r[b]=r[c]=r[d]=r[e]=2, r[f]=1, r[g]=0

  (3)p[a]=e, p[b]=f, p[c]=p[d]=p[e]=p[f]=p[g]=g
  r[a]=4, r[b]=4, r[c]=4, r[d]=3, r[e]=2, r[f]=1, r[g]=0

  (4)p[a]=p[b]=p[c]=p[d]=p[e]=p[f]=p[g]=g
  r[a]=6, r[b]=5, r[c]=4, r[d]=3, r[e]=2, r[f]=1, r[g]=0



表序问题

- 教材P200算法7.10

(1) par-do：初始化p[k]和distance[k] //O(1)

(2)执行 $\lceil \log n \rceil$ 次 //O(logn)

(2.1) for all k par-do //O(1)

如果k的后继不等于k的后继之后继，则

(i) distance[k]= distance[k]+ distance[p[k]]

(ii) p[k]=p[p[k]]

(2.2) for all k par-do

rank[k]=distance[k] //O(1)

运行时间：t(n)=O(logn) p(n)=n

求森林的根

- 问题描述

一组有向树F中, 如果<i, j>是F中的一条弧， 则p[i]=j(即j是i的双亲)；若i为根，则p[i]=i。求每个结点j(j=1~n)的树根s[j].

- 示例

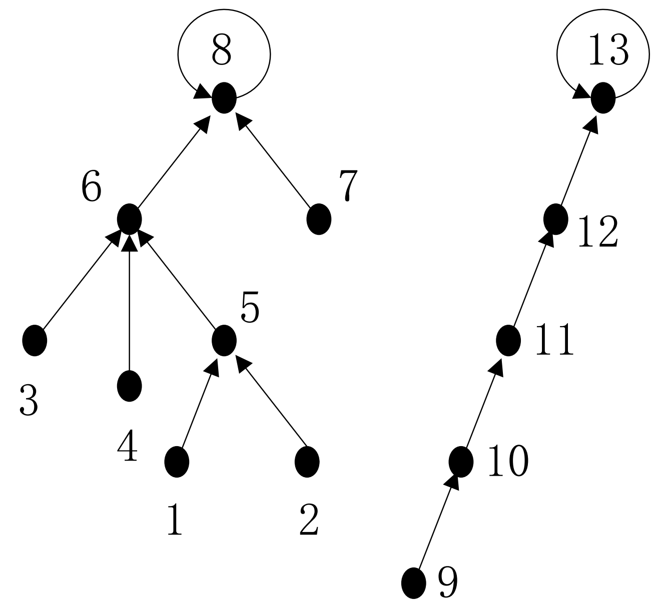
初始时

p[1]=p[2]=5　p[3]=p[4]=p[5]=6

p[6]=p[7]=8　p[8]=8　P[9]=10

p[10]=11 p[11]=12　p[12]=13　p[13]=13

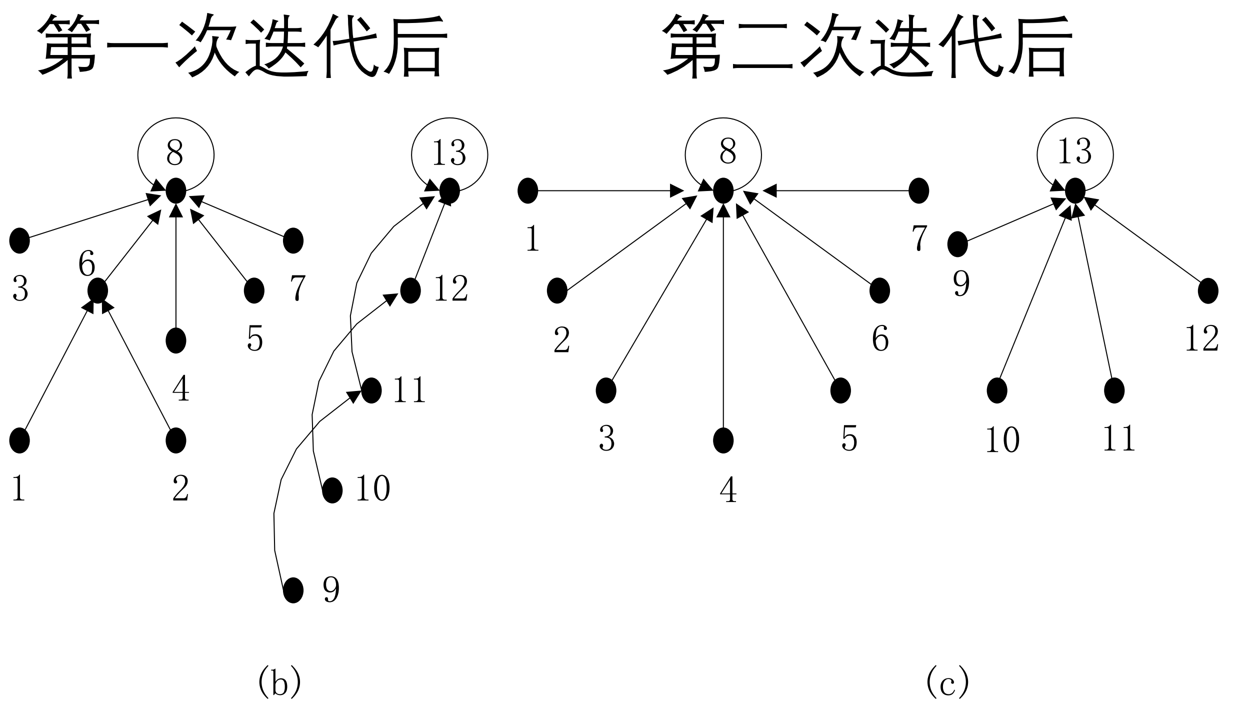
s[i]=p[i]



(a)

求森林的根

- 示例



- 算法：教材P202算法7.11
- 运行时间：t(n)=O(logn)   W(n)=O(nlogn)

第七章 并行算法常用设计技术

- 7.1 划分设计技术
- 7.2 分治设计技术
- 7.3 平衡树设计技术
- 7.4 倍增设计技术
- **7.5 流水线设计技术**

流水线设计技术

- 设计思想
  - 将算法流程划分成p个前后衔接的任务片断，每个任务片断的输出作为下一个任务片断的输入;
  - 所有任务片断按同样的速率产生出结果。
- 评注
  - 流水线技术是一种广泛应用在并行处理中的技术;
  - 脉动算法(Systolic algorithm)是其中一种流水线技术;
  - 近年最典型的软件流水线应用：DNN training

5-point DFT的计算

- n点离散傅里叶变换将给定的序列（$a_0, a_1, \ldots a_{n-1}$）变换为序列（ $b_0, b_1, \ldots b_{n-1}$ ）：

$$b_j = \sum_{k=0}^{n-1} a_k \omega^{kj}, 0 \le j \le n-1$$

其中$\omega = e^{2\pi i/n}$

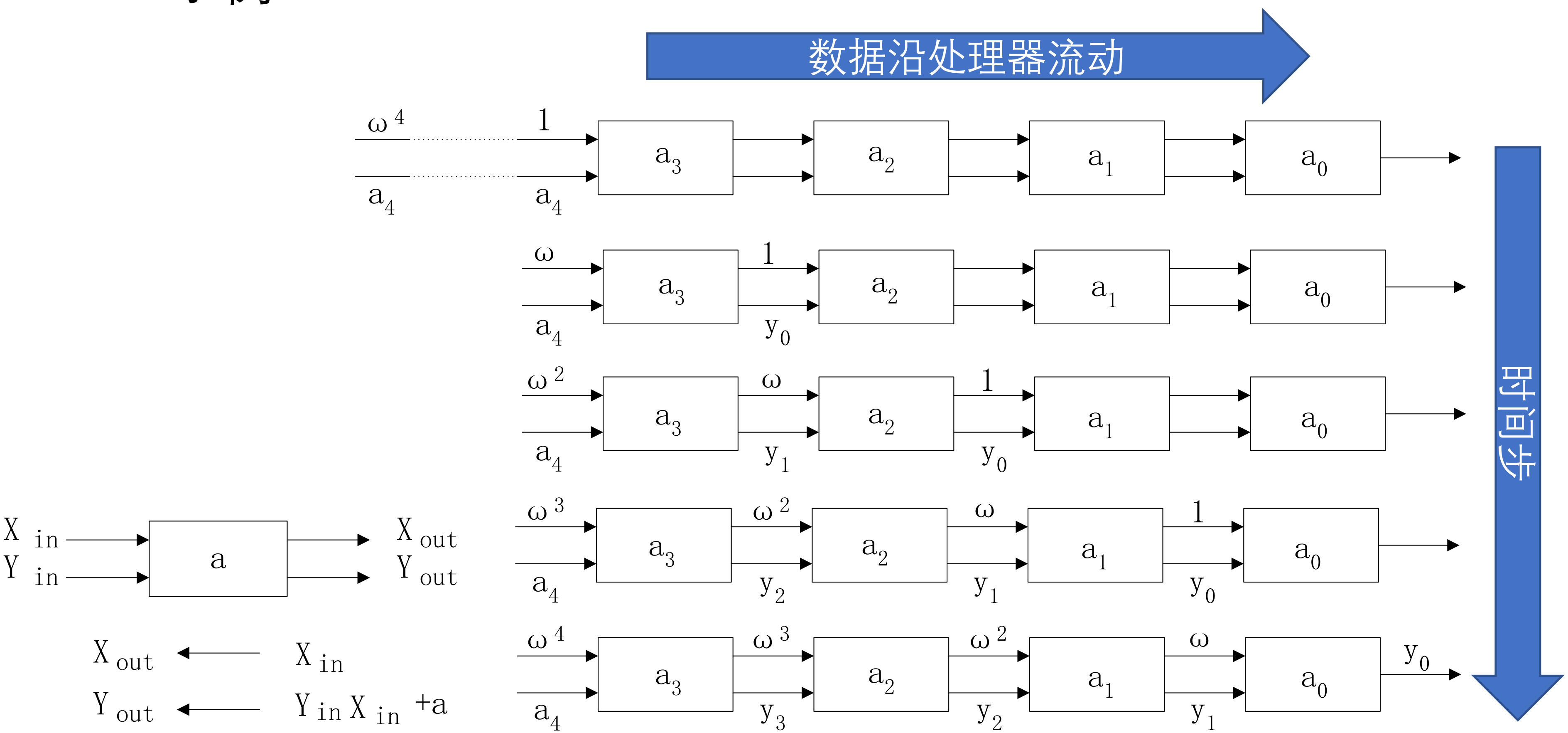
5-point DFT的计算

- 5-point DFT的计算。应用Horner法则：

$$\begin{cases} y_0 = b_0 = a_4\omega^0 + a_3\omega^0 + a_2\omega^0 + a_1\omega^0 + a_0 \\ y_1 = b_1 = a_4\omega^4 + a_3\omega^3 + a_2\omega^2 + a_1\omega^1 + a_0 \\ y_2 = b_2 = a_4\omega^8 + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \\ y_3 = b_3 = a_4\omega^{12} + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \\ y_4 = b_4 = a_4\omega^{16} + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \end{cases} \Rightarrow \begin{cases} y_0 = (((a_4\omega^0 + a_3)\omega^0 + a_2)\omega^0 + a_1)\omega^0 + a_0 \\ y_1 = (((a_4\omega^1 + a_3)\omega^1 + a_2)\omega^1 + a_1)\omega^1 + a_0 \\ y_2 = (((a_4\omega^2 + a_3)\omega^2 + a_2)\omega^2 + a_1)\omega^2 + a_0 \\ y_3 = (((a_4\omega^3 + a_3)\omega^3 + a_2)\omega^3 + a_1)\omega^3 + a_0 \\ y_4 = (((a_4\omega^4 + a_3)\omega^4 + a_2)\omega^4 + a_1)\omega^4 + a_0 \end{cases}$$

5-point DFT的计算

- 示例:



(a) (b)

二维求和阵列

- 用多线程流水方法计算下面阵列

$$A:\quad \begin{pmatrix} 0 & 1 & 3 & 6 & 10 & 15 & 21 \\ 1 & 2 & 5 & 11 & 21 & 36 & 57 \\ 3 & 5 & 10 & 21 & 42 & 78 & 135 \\ 6 & 11 & 21 & 42 & 84 & 162 & 297 \end{pmatrix}$$

计算公式如下：

$$\mathrm{A}[i,j]=\begin{cases} 0 & if\ i=0\ and\ j=0 \\ A[0,j-1]+j & if\ i=0\ and\ j>0 \\ A[i-1,0]+i & if\ i>0\ and\ j=0 \\ \mathrm{A}[i-1,j]+\mathrm{A}[i,j-1] & other \end{cases}$$

二维求和阵列

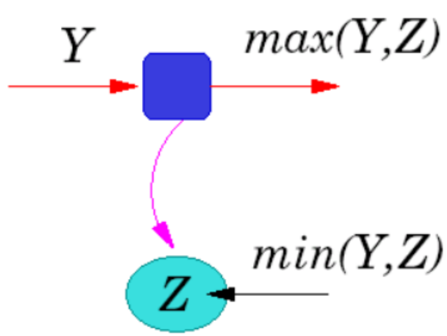
线程 \步骤	$S0$	$S1$	$S2$	$S3$	$S4$	$S5$	$S6$			
$T0$	0	1	3	6	10	15	21			
$T1$		1	2	5	11	21	36	57		
$T2$			3	5	10	21	42	78	135	
$T3$				6	11	21	42	84	162	297

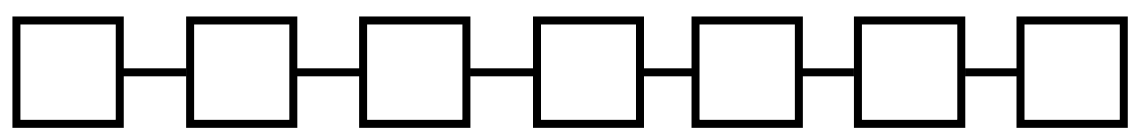
计算公式：

$$\mathrm{data}[T_i, S_j] = \begin{cases} 0 & if\ i=0\ and\ j=0 \\ \mathrm{data}[T_0, S_{j\text{-}1}] + j & if\ i=0\ and\ j>0 \\ \mathrm{data}[T_{i\text{-}1}, S_0] + i & if\ i>0\ and\ j=0 \\ \mathrm{data}[T_{i\text{-}1}, S_j] + \mathrm{data}[T_i, S_{j\text{-}1}] & other \end{cases}$$

Systolic排序算法

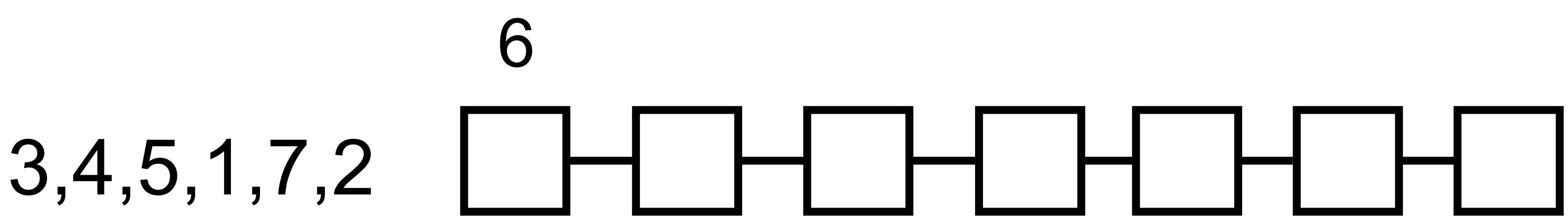
- 一维线性阵列上的心动排序算法
  - 每个处理器存储自己见过的最小数字
  - 每个处理器初始存储值为无穷大
  - 每当从左邻居接收到Y，留下Y和自己存储的Z中的较小值，将较大值传给右邻居。





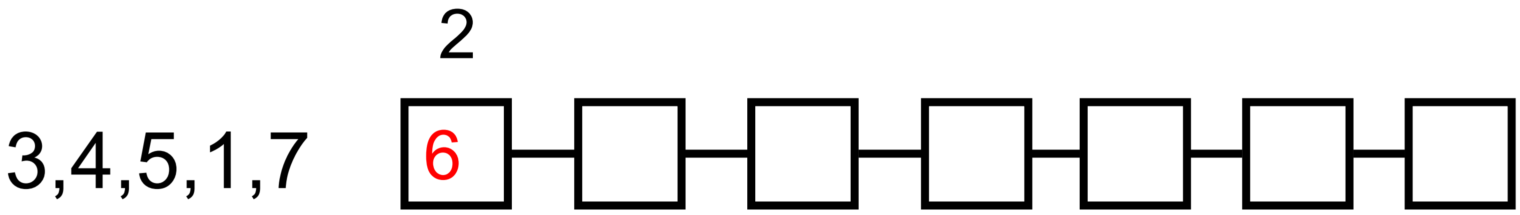
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



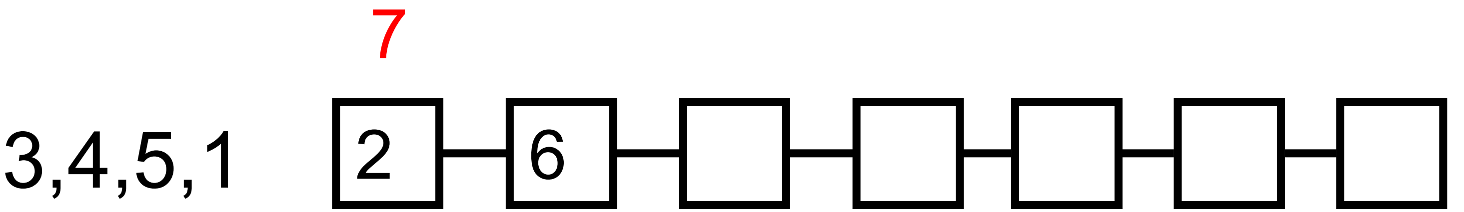
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



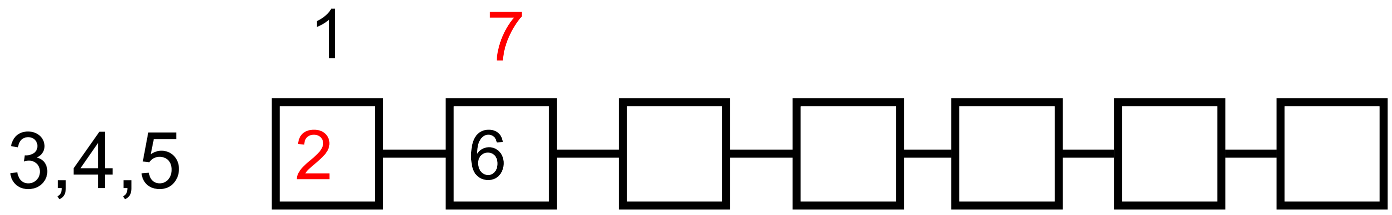
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



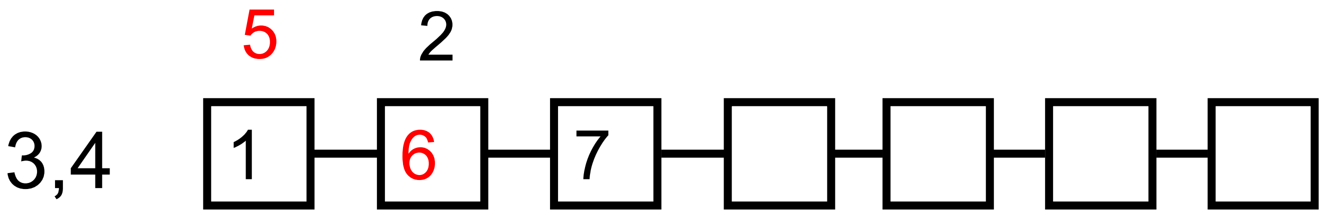
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



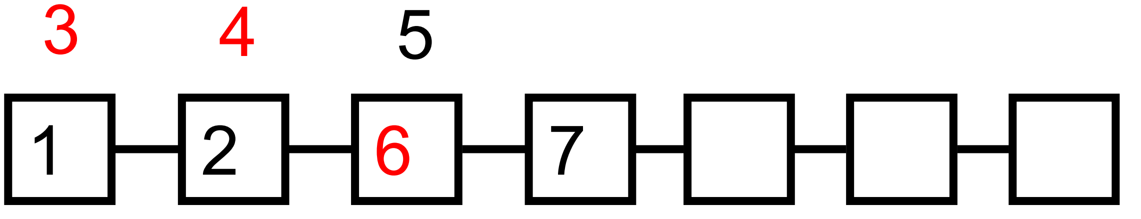
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



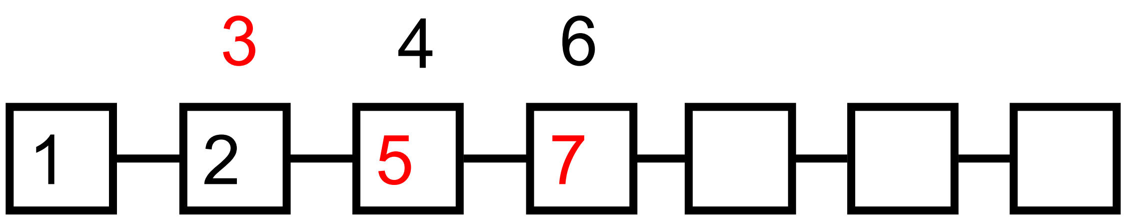
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



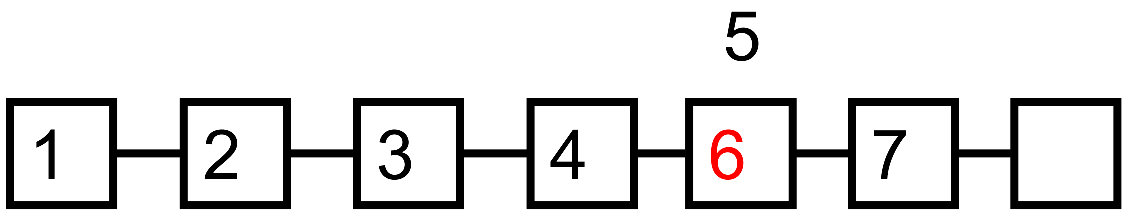
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



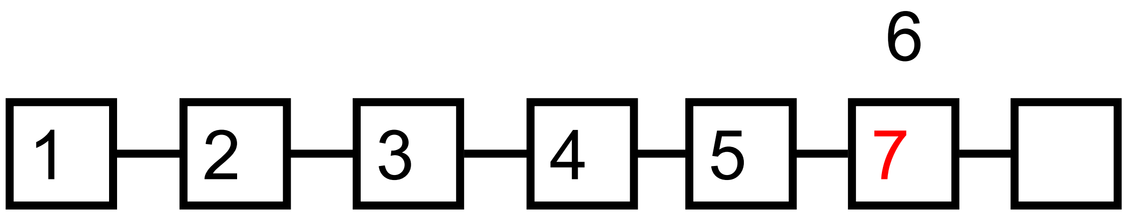
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6



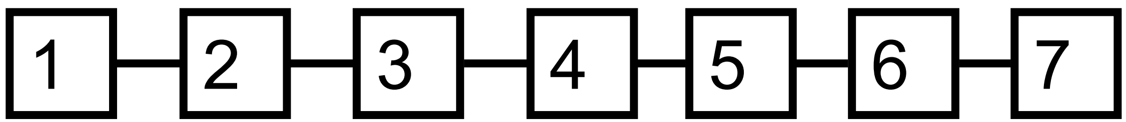
Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6

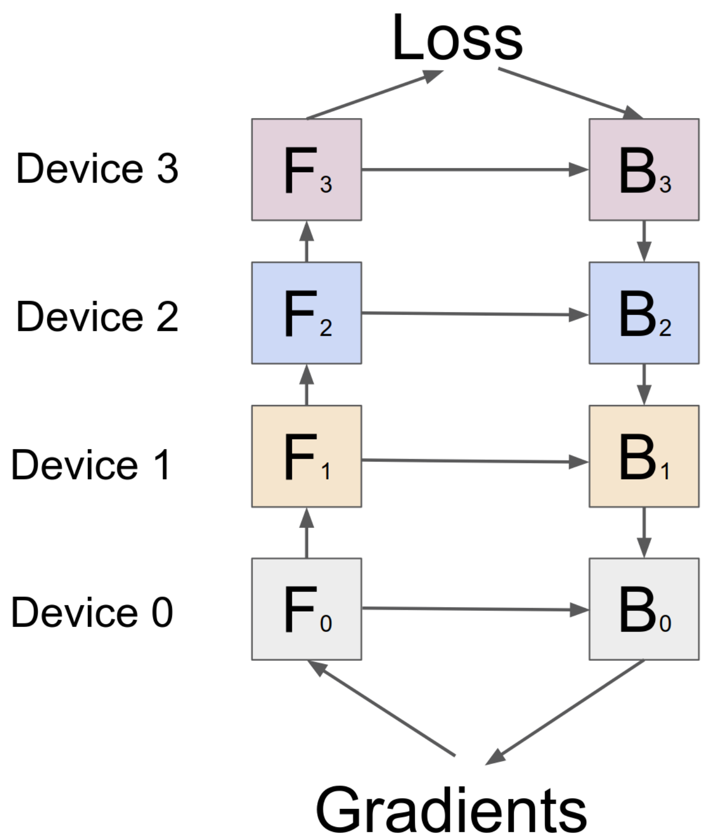


Systolic排序算法

- 示例: Sorting 3, 4, 5, 1, 7, 2, 6

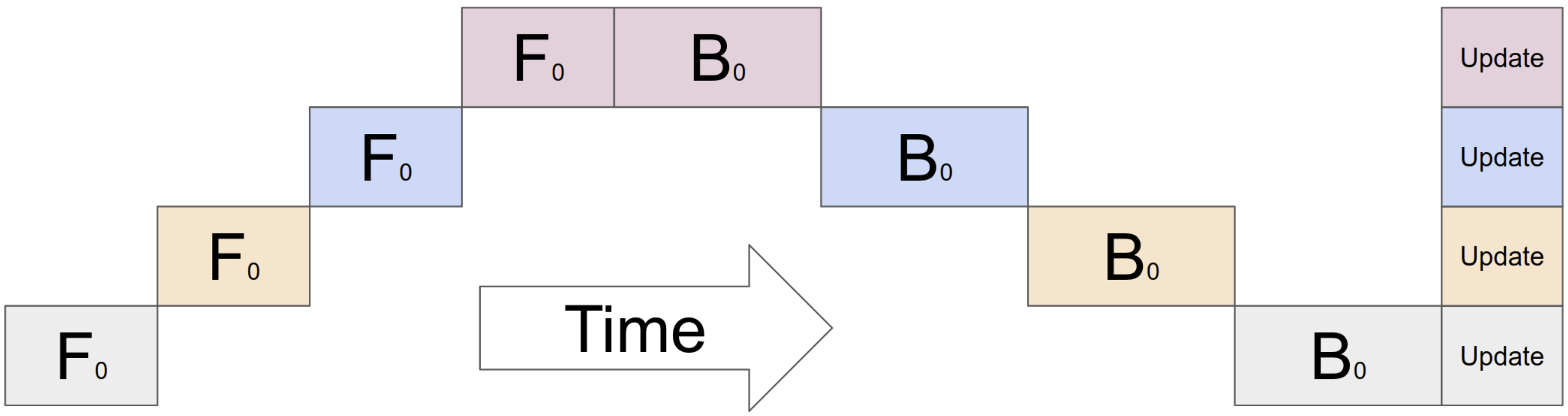


Pipeline parallelism for DNN



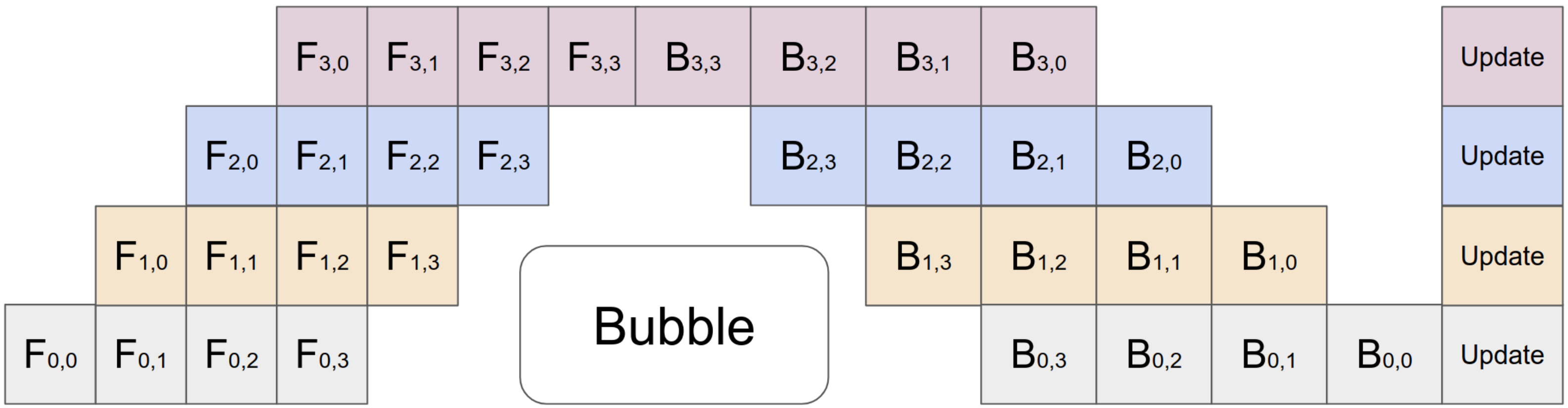
DNN训练过程中前向与后向传播的数据依赖

Pipeline parallelism for DNN



每个设备负责一层，由于数据依赖导致无法并行

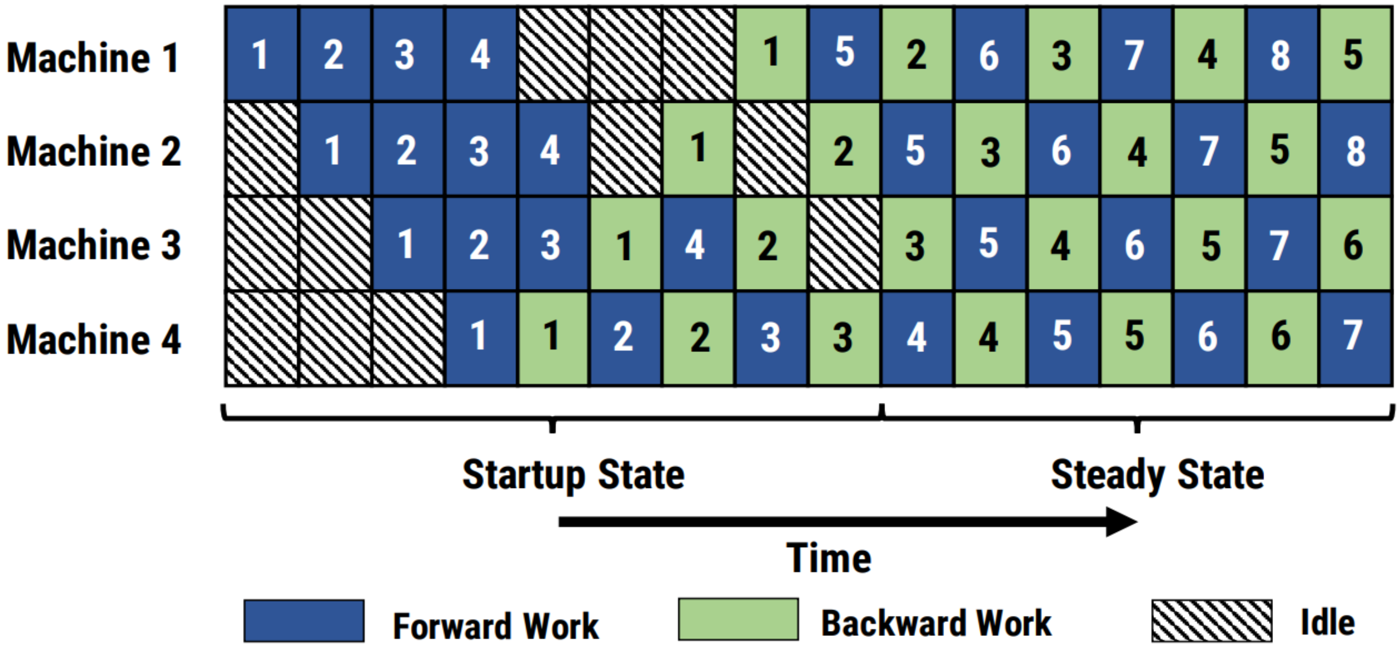
Pipeline parallelism for DNN



将每一层划分为更小的任务执行单元，获得流水线并行加速，但仍有不少气泡

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, Zhifeng Chen: GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. NeurIPS 2019: 103-112

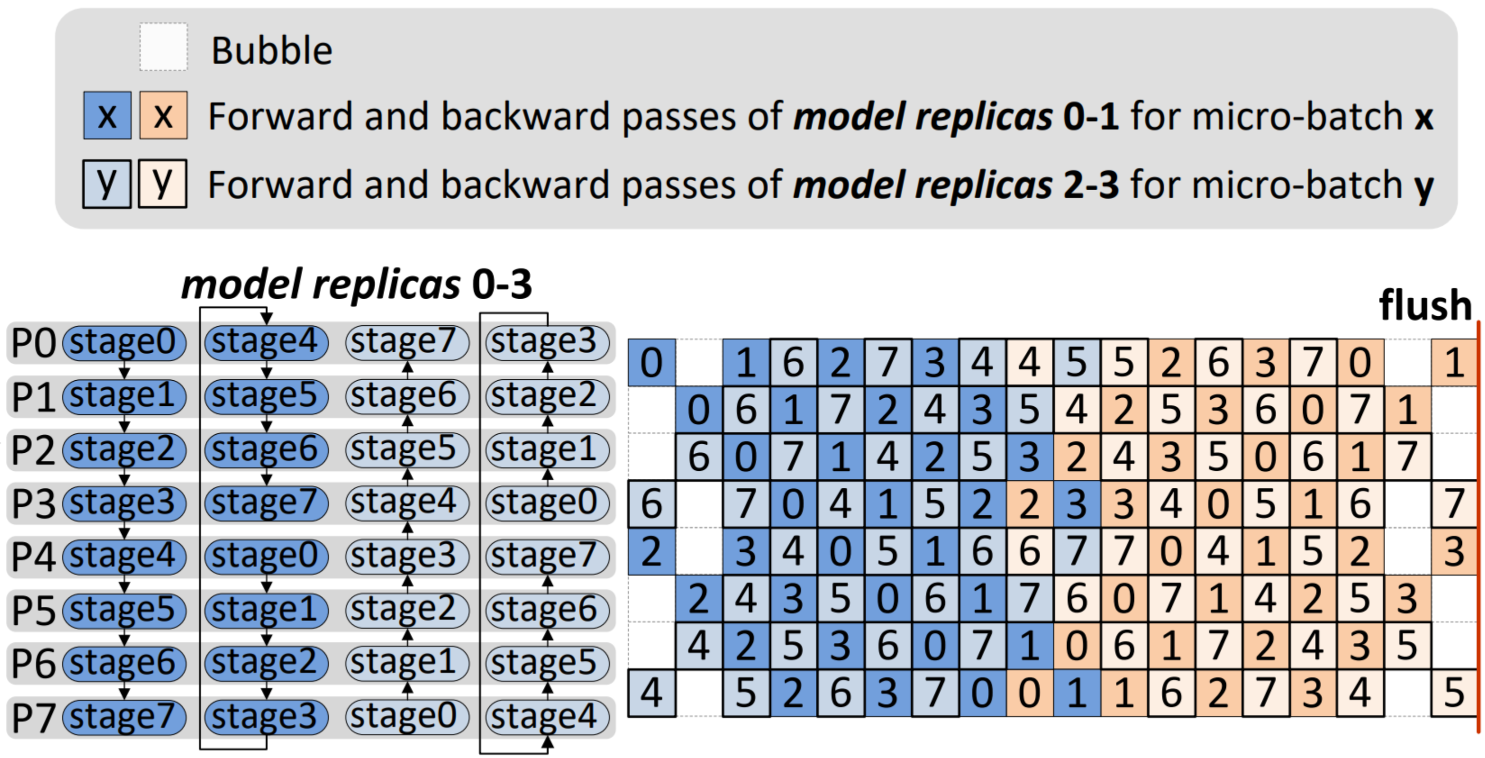
Pipeline parallelism for DNN



一种紧凑的Pipeline设计

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, Matei Zaharia: PipeDream: generalized pipeline parallelism for DNN training. SOSP 2019: 1-15

Pipeline parallelism for DNN



一种更紧凑的Pipeline设计

Shigang Li, Torsten Hoefler: Chimera: efficiently training large-scale neural networks with bidirectional pipelines. SC 2021: 27