# 并行计算作业 存档

傅申 PB20000051

**课件 5.1.** 改写求最大值问题的并行算法, 要求不使用数组 `M`.

```
1  begin
2    for 1 ⩽ i ⩽ p par-do
3      B[i, 1] = 1
4    end for
5    for 1 ⩽ i, j ⩽ p par-do
6      if B[i, 1] = 1 and A[i] > A[j] then
7        B[i, 1] = 0
8      end if
9    end for
10 end
```

**教材习题 5.6.** APRAM 上求和算法

1. 全局读写时间为 $d$, 同步障时间为 $B(p)$.
   (1) $O\left(\frac{n}{p}+d\right)$
   (2) $B(p)$
   (3) $\lceil\log_B(p(B-1))+1\rceil-1$ 次迭代
      (3.1) $O(Bd)$
      (3.2) $B(p)$

   总共 $O\left((Bd+B(p))\log_B p+\frac{n}{p}\right)$.
2. Barrier 语句确保每个处理器计算完局和并写入 SM 后, 局和才被读取, 避免脏读.

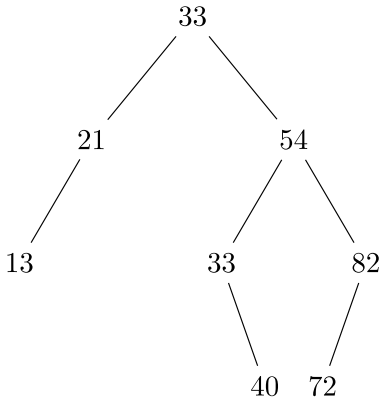
**教材习题 5.7.** BSP 上求和算法

1. 忽略传输建立时间, 同步障的时间显然是 $O(L)$ 的.
   (1) $O\left(\frac{n}{p}+g\right)$
   (2) $O(L)$
   (3) $\lceil\log_d(p(d-1))+1\rceil-1$ 次迭代
      (3.1) $O(g(d+1)+d)=O(gd)$
      (3.2) $O(L)$

   总共 $O\left((gd+L)\log_d p+\frac{n}{p}\right)$.
2. 首先 BSP 模型的一个超级步中一个处理器最多可以传送 $h$ 条消息 ($L\geq gh$), 而在 (3.1) 中, 一个处理器要发送/接收 $d+1$ 条消息, 因此 $d$ 要满足 $d\leq h-1$; 其次, 考虑到时间性能, 应选择使时间性能尽可能好, 同步障时间尽可能小的 $d\leq h-1$ 值.

**教材习题 6.3.** 构造 PRAM-CRCW 模型上执行快排序所用的二叉树.

假设总是编号较小的处理器写入变量, 则二叉树为:




- $P_0$ 能成功写入 $root$, 因此树根为 33.
  - $P_1$ 能成功写入 $LC_0$, 因此 31 为根的左节点. $P_0$ 的左子树中还有 [13].
    - $P_2$ 写入 $LC_1$, 因此 13 为 31 的左节点.
  - $P_3$ 能成功写入 $RC_0$, 因此 54 为根的右节点. $P_0$ 的右子树里还有 [82, 33, 40, 72].
    - $P_5$ 能成功写入 $LC_3$, 因此 33 为 54 的左节点. $P_3$ 的左子树里还有 [40].
      - $P_6$ 写入 $RC_5$, 因此 40 为 33 的右节点.
    - $P_4$ 能成功写入 $RC_3$, 因此 82 为 54 的右节点. $P_3$ 的右子树里还有 [72].
      - $P_7$ 写入 $LC_4$, 因此 72 为 82 的左节点.

**课件 6.1.** 如何实现算法 6.3 所构造二叉排序树并行地转化为有序数组, 写出实现的并行算法, 其时间复杂度和并行计算模型各是什么?

采用类似顺序统计量的做法. 算法见下一页, 各个部分的功能为:

(1) 初始化, $O(1)$.
  (1.1) 初始化 size 数组 `s`, 其中叶节点初始化为 1, 否则为 0;
  (1.2) 初始化 rank 数组 `r`, `r[root] = -2` 是为了使 (4) 能从根节点开始.
  (1.3) `s[n + 1] = 0` 是为了方便 (2.1) 中的赋值;
  (1.4) `r[n + 1]` 是冗余量, 方便 (3.2.2) 中的赋值.
(2) 从下到上计算各个节点的 size `s[i]`, 直到根节点的 size 更新, $O(\log n)$.
  (2.1) 若节点的 size 为 0 且孩子节点的 size 已更新, 则更新节点 size.
(3) 从上到下计算各个节点的 rank `r[i]`, $O(\log n)$.
  (3.1) 初始化 `updated`, 它是该轮迭代中是否有 `r[i]` 被更新的标记;
  (3.2) 如果 `r[i] = -2`, 则它的父节点在上一轮迭代中被更新;
    (3.2.1) 计算并更新 `r[i]`;
    (3.2.2) 标记其子节点为 -2 并更新 `updated`, 其中 `r[n + 1]` 作为冗余量永远不会被处理, 因此不用考虑没有子节点的情况.
(4) 按照 `r[i]` 的值将排序好的数组写入 `B` 中, $O(1)$.

总时间为 $O(\log n)$. 因为存在同时读写的情况, 所以模型为 PRAM-CRCW.

```
begin
  (1) for each processor i par-do
      (1.1) if LC[i] = RC[i] = n + 1 then s[i] = 1 else s[i] = 0 end if
      (1.2) if i = root then r[i] = -2 else r[i] = -1 end if
      (1.3) s[n + 1] = 0
      (1.4) r[n + 1] = 0
      end for
  (2) repeat for each processor i do
      (2.1) if s[i] = 0
               and (LC[i] = n + 1 or s[LC[i]] ≠ 0)
               and (RC[i] = n + 1 or s[RC[i]] ≠ 0) then
             s[i] = 1 + s[LC[i]] + s[RC[i]]
           end if
      until s[root] ≠ 0
  (3) repeat for each processor i do
      (3.1) updated = false
      (3.2) if r[i] = -2 then
           (3.2.1) if i = root then
                     r[i] = s[LC[i]]
                   else if i = LC[f[i]] then
                     r[i] = r[f[i]] - s[i] + s[RC[i]]
                   else
                     r[i] = r[f[i]] + s[LC[i]] + 1
                   end if
           (3.2.2) r[LC[i]] = -2
                   r[RC[i]] = -2
                   updated = true
           end if
      until updated = false
  (4) for each processor i par-do
        B[r[i]] = A[i]
      end for
end
```

**课件 Activity 11.** 在 PRAM-CREW 模型上, 用 $n$ 个处理器在 $O(1)$ 时间内求出数组 $A[1..n] = \{0, \cdots, 0, 1, \cdots, 1\}$, 最先为 1 值的下标. 写出并行伪代码。

```
begin
  for each processor i < n par-do
    if A[i] = 0 and A[i + 1] = 1 then
      index = i + 1
    end if
  end for
end
```

**教材习题 7.3.** 分析算法 7.3 的复杂度, 并手动运行.

(1) 总共 $O(\log m + n)$:

(1) 显然时间是 $O(1)$ 的;

(2) 求 $rank$ 可以使用二分查找, $O(\log n)$;

(3) 将数据拷贝存储, 时间为 $O(\log m + n)$.

(2) 算法执行得到: $rank(b_4 : A) = 3$, $rank(b_8 : A) = 6$, $rank(b_{12} : A) = 10$. 划分得:

| $B$ | 3, 4, 5, 6 | 8, 10, 12, 13 | 14, 15, 20, 21 | 22, 26, 29, 31 |
|---|---|---|---|---|
| $A$ | 0, 1, 2 | 7, 9, 11 | 16, 17, 18, 19 | 23, 24, 25, 27, 28, 30, 33, 34 |
| 归并 | 0, 1, 2, 3, 4, 5, 6 | 7, 8, 9, 10, 11, 12, 13 | 14, 15, 16, 17, 18, 19, 20, 21 | 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34 |

最后归并得到 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34).

**教材习题 7.6.** 分析算法 7.9 的总运算量, 并手动运行.

(1) $W(n) = O(n)$

(1) $O(n)$

(2) $\sum_{h=1}^{\log n} O\left(\frac{n}{2^h}\right) = O(n)$

(3) $\sum_{h=0}^{\log n} O\left(\frac{n}{2^h}\right) = O(n)$

(2) 如下, 其中计算的是前缀“和”

(1) 初始化, $B$ 如下:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

(2) 正向遍历, $B$ 如下:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 3 | 7 | 11 | 15 | | | | |
| 10 | 26 | | | | | | |
| 36 | | | | | | | |

(3) 反向遍历, $C$ 如下:

| 1 | 3 | 6 | 10 | 15 | 21 | 28 | 36 |
|---|---|---|---|---|---|---|---|
| 3 | 10 | 21 | 36 | | | | |
| 10 | 36 | | | | | | |
| 36 | | | | | | | |

第一行 $C(0)$ 就是前缀和.

**教材习题 2.6.** 分析洗牌交换网络.

**节点度** 4, 包括交换的 2 个度和洗牌的两个度;
**网络直径** 对图中 $N=8$ 的洗牌交换网络, 节点 0 和 7 之间的路径是最长的, 距离为 5, 最短路径为 $0\to1\to2\to3\to6\to7$, 因此网络直径为 5; 对 $N=2^n$ 的洗牌交换网络, 仍然是节点 0 和 $2^n-1$ 之间的路径最长, 最短路径为交换/洗牌轮流进行的路径, 即 $0\to1\to...\to\left(2^i-2\right)\to\left(2^i-1\right)\to...\to(2^n-2)\to(2^n-1)$, 长度 (网络直径) 为 $2n-1$.
**网络对剖宽度** 对图中 $N=8$ 的洗牌交换网络, 将节点分为 $\{0,1,2,4\}$ 和 $\{3,5,6,7\}$ 两部分, 需要移去的边只有 2 条 $(2\leftrightarrow3,4\leftrightarrow5)$ 为最小值, 因此对剖宽度为 2; 对 $N=2^n$ 的情况, 对剖宽度有上界 $O\left(\frac{N}{n}\right)$, 如下表, 证明见 MIT 的 Theory of Parallel Systems (SMA 5509) 课程的 Lecture 18 的 Note 最后一章.

| $N$ | 对剖宽度 | 去掉的边 |
|---|---|---|
| 2 | 1 | $0\leftrightarrow1$ |
| 4 | 1 | $1\leftrightarrow2$ |
| 8 | 2 | $2\leftrightarrow3,4\leftrightarrow5$ |
| 16 | 4 | $2\leftrightarrow3,8\leftrightarrow9,10\leftrightarrow11,14\leftrightarrow15$ |

**教材习题 2.7.** 分析蝶形网络.

**节点度** 行 0 和行 $k$ 的节点度为 2, 中间行的节点度为 4.
**网络直径** 直径为 $2k$, 对应的路径为行 0 的第一个节点到最后一个节点的路径, 需要先到行 $k$ 再回到行 0.
**网络对剖宽度** 从中间对剖下去, 需要去掉 $2\times2^{k-1}=2^k$ 条边, 即为对剖宽度.

**教材习题 2.15.** 分析在超立方上单点散播的通信时间.

如图中过程所示, 每步传输都是传播一条, 每次传递的信包量都会减半, 而涉及的节点数量都会加倍, 因此只需要 $\lg p$ 步传输即可完成单点散播. 因此, SF 方式的传输时间为

$$\begin{aligned}t_{\text{one-to-all-pers}}(\text{SF})&=\sum_{i=1}^{\lg p}\left(t_s+\left(\frac{mp}{2^i}t_w+t_h\right)l\right)\\&=t_s\lg p+\sum_{i=1}^{\lg p}\frac{mp}{2^i}t_w\qquad(\textbf{忽略 }t_h\textbf{ 并且 }l=1)\\&=t_s\lg p+mt_w(p-1)\end{aligned}$$

同理, 对于 CT 方式, 传输时间为

$$
\begin{aligned}
t_{\text{one-to-all-pers}}(\mathrm{CT}) &= \sum_{i=1}^{\lg p}\left(t_s + \frac{mp}{2^i}t_w + lt_h\right) \\
&= t_s \lg p + \sum_{i=1}^{\lg p}\frac{mp}{2^i}t_w \qquad (\textbf{忽略 } t_h \textbf{ 并且 } l=1) \\
&= t_s \lg p + mt_w(p-1)
\end{aligned}
$$

因此, 两种方式的传输时间相同, 均为 $t_{\text{one-to-all-pers}} = t_s \lg p + mt_w(p-1)$.

**Sch9.S1.** 试将 Cannon 分块乘法算法 9.5 改为共享存储 PRAM-EREW 模型上的算法, 并分析时间复杂度.

如下

```
1  begin
2    (1) for all P[i, j] par-do
3        (1.1) C[i, j] = 0
4        (1.2) for k = 0 to √p - 1 do
5              (1.2.1) C[i, j] += A[i, (i + j + k) % √p] * B[(i + j + k) % √p, j]
6              end for
7        end for
8  end
```

其中 (1.1) 的时间为 $O(1)$, 而 (1.2) 执行了 $\sqrt{p}$ 次 $\left(n \,/\, \sqrt{p}\right) \times \left(n \,/\, \sqrt{p}\right)$ 子块乘法, 时间为 $\sqrt{p} \times \left(n \,/\, \sqrt{p}\right)^3 = n^3 \,/\, p$, 因此时间复杂度为 $O(n^3 \,/\, p)$.

**教材习题 9.9.** 分析算法 9.7 的运行时间 ($n^2$ 个处理器的并行系统上用 PRAM-CREW 模型施行两个 $n \times n$ 矩阵相乘的算法)

(2.1) 的运行时间为 $t_a$, (2.2) 迭代了 $n$ 轮, 其中每一轮需要读存储器 3 次, 进行一次乘法和一次加法, 再写入存储器一次, 每一轮时间为 $4t_a + 2t_c$, 因此总的并行运行时间为 $(4n+1)t_a + 2nt_c$.