第四章

4.5 假定 $P_i(1 \leqslant i \leqslant n)$开始时存有数据 d_i，所谓累加求和系指用 $\sum_{j=1}^{i} d_j$来代替 P_i 中的原始值 d_i。算法 4.3 给出了在 PRAM 模型上累加求和算法：

算法 4.3　PRAM-EREW 上累加求和算法

输入： P_i 中保存有 $d_i, l \leqslant i \leqslant n$

输出： P_i 中的内容为 $\sum_{j=l}^{i} d_j$

```
Begin
  for j = 0 to log n − 1 do
    for i = 2^j + 1 to n par-do
      (i) P_i = d_{i−2^j}
      (ii) d_i = d_i + d_{i−2^j}
    endfor
  endfor
End
```

(1)试用 $n=8$ 为例，按照上述算法逐步计算出累加和。

(2)分析算法 4.3 的时间复杂度。

4.6 在 APRAM 模型上设计算时，应尽量使各处理器内的局部计算法时间和读写时间大致与同步时间 B 相当。当在 APRAM 上计算 n 个数的和时，可以借用 B 叉树求和的办法。

假定有 p 个处理器计算 n 个数的和，此时每个处理器上分配 n/p 个数，各处理器先求出自身的局和；然后从共享存储器中读取它的 B 个孩子的局和，累加后置入指定的共享存储单元 SM 中；最后根处理器所计算的和即为全和。算法 4.4 示出了 APRAM 上的求和算法：

算法 4.4　APRAM 上求和算法

输入： n 个待求和的数

输出： 总和在共享存储单元 SM 中

```
Begin
  (1) 各处理器求 n/p 个数的局和，并将其写入 SM 中
  (2)Barrier
  (3) for k = ⌈log_B(p(B − 1) + 1)⌉ − 2 downto 0 do
        (3.1) for all P_i, 0 ≤ i ≤ p − 1, do
              if P_i 在第 k 级 then
                P_i 计算其 B 个孩子的局和并与其自身局和相加，然后将结果写入 SM
                中
              endif
            end for
        (3.2) Barrier
      end for
```

End

(1)试用 APRAM 模型之参数,写出算法的时间复杂度函数表达式。

(2)试解释 Barrier 语句的作用。

4.7 欲在 BSP 模型上计算 n 个数的和,可以在 d 叉树上进行。假定用 p 个处理器求 n 个数的和,则每个处理器分配有 n/p 个数。首先,各处理器求 n/p 个数的局和;然后在 d 叉树上自下而上求全和,其全过程如算法 4.5 所示。

算法 4.5 BSP 上求和算法

输入: n 个待求和的数

输出: 总和在根处理器 P_0 中

Begin

(1) **for all** P_i, $0 \leqslant i \leqslant p-1$ **do** /* 各处理器求各自局和 */

(1.1) P_i 计算 n/p 个数的局和

(1.2) **if** P_i 在第 $\lceil \log_B(p(B-1)+1) \rceil - 1$ 级 **then**

P_i 将其局和发往父节点

endif

endfor

(2) **Barrier**

(3) **for** $k = \lceil \log_B(p(B-1)+1) \rceil - 2$ **downto** 0 **do** /* 上播并求和 */

(3.1) **for all** P_i, $0 \leqslant i \leqslant p-1$ **do**

if P_i 在第 r 级 **then**

P_i 接收 d 个孩子消息,并将它们与其本身局和相加;然后将结果发往父节点

endif

endfor

(3.2) **Barrier**

endfor

End

(1)试分析算法 4.5 的时间复杂度。

(2)d 值如何确定?

4.8 在给定时间 t 内,尽可能多的计算输入值的和也是一个求和问题。如果在 logP 模型上求此问题时,要是 $t < L + 2 \cdot o$,则在一个单处理机上即可最快地完成;要是 $t > L + 2 \cdot o$ 时,则根处理器应在 $t-1$ 时间完成局和的接收工作,然后用一个单位时间完成加运算而得最终的全和。而根的远程子节点应在 $(t-1)-(L+2 \cdot o)$ 时刻开始发送数据,其兄妹子节点应依次在 $(t-1)-(L+2 \cdot o+g)$, $(t-1)-(L+2 \cdot o+2g)$, … 时刻开始发送数据。图 4.4 示出了 $t=28$, $p=8$, $L=5$, $o=2$, $g=4$ 的 logP 模型上的通信(即发送/接收)调度树。试分析此通信调度树的工作原理和图中节点中的数值是如何计算的?

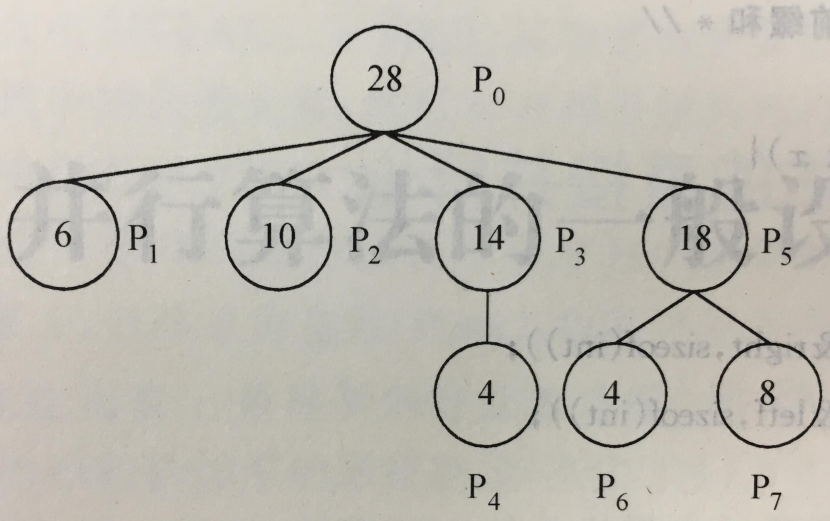

图 4.4　$t=28, p=8, L=5, o=2, g=4$ 的通信调度树

4.10　试分析如下用 BSPLib 并行求 4 个整数 1,2,3,4 的前缀和的过程(参见图 4.6)。

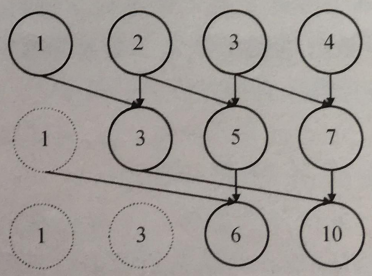

图 4.6　求整数 1,2,3,4 的前缀和过程

```
//* 用 BSPLib 求前缀和 *//
#include "bsp.h"
int bsp_allsums(int x){
  int i,left,right;

  bsp_push_reg(&right,sizeof(int));
  bsp_push_reg(&letf,sizeof(int));
  bsp_sync();

  right=x
  for(i=1;i<bsp_nprocs();i*=2){
    if (bsp_pid()+i<bsp_nprocs())
      bsp_put(bsp_pid()+i,&right,&left,0,sizeof(int));
    bsp_sync();
    if (bsp_pid()>=i) right=left+right;
  }
  bsp_pop_reg(&right);
  bsp_pop_reg(&left);
  return right;
}
void main(){
  bsp_begin(bsp_nprocs());
    printf("On %d sum is %d\n",bsp_pid(),bsp_allsums(1+bsp_pid()));
bsp_end();
}
```