第四章

4.5

(1) n=8 时的计算过程

j=0; i =1 to 8

$P_1=d_1$, $P_2=d_1+d_2$, $P_3=d_2+d_3$, $P_4=d_3+d_4$, $P_5=d_4+d_5$, $P_6=d_5+d_6$, $P_7=d_6+d_7$, $P_8=d_7+d_8$

j=1; i =3 to 8

$P_3=d_1+(d_2+d_3)$, $P_4=(d_1+d_2)+(d_3+d_4)$, $P_5=(d_2+d_3)+(d_4+d_5)$,

$P_6=(d_3+d_4)+(d_5+d_6)$, $P_7=(d_4+d_5)+(d_6+d_7)$, $P_8=(d_5+d_6)+(d_7+d_8)$

j=2; i = 5 to 8

$P_5=d1+((d_2+d_3)+(d_4+d_5))$, $p6=(d1+d2)+((d_3+d_4)+(d_5+d_6))$,

$P_7=(d_1+(d_2+d_3))+((d_4+d_5)+(d_6+d_7))$, $p8=((d_1+d_2)+(d_3+d_4))+((d_5+d_6)+(d_7+d_8))$

(2) 时间复杂度: $O(\log n)$

4.6

(1) 设APRAM 同步障时间为B，全局读写时间为d，本地读写时间为单位时间。

算法(1): n/p-1+d

算法(2): B

算法(3.1): B+1+d

算法(3.2): B

因此算法的时间复杂度为:

$$\begin{aligned}T &= n/p-1+d+B+k(2B+d+1)\\ &= n/p-1+d+B+(2B+d+1)(\lceil\log_B(p(B-1)+1)\rceil-2)\\ &= O(n/p+(2B+d+1)\lceil\log_B(p(B-1)+1)\rceil\end{aligned}$$

(2) Barrie的意思是设置同步障，所有处理器在该处均需等待别的处理器到达后才能继续执行下一个语句。

4.7

(1) 时间复杂度:

设同步障时间为B, 传输一个消息时间为g, 本地计算一次操作的时间为单位时间.

(1) —— (1.1) n/p – 1 (1.2) g

(2) B

(3) —— (3.1)g*d+(d+1)+g (3.2) B

T=n/p-1+g+B+(k+1)(g*d+d+1+g+B)

$=O(n/p+g+B-1+(\lceil\log_B(p(B-1)+1)\rceil-2+1)(g*d+d+1+g+B))$

$= O(n/p+(g+B-1)+(gd+d+g+1+B)(\lceil\log_B(p(B-1)+1)\rceil-1))$

$=O(n/p-(gd+d+2)+(gd+d+g+1+B)\lceil\log_B(p(B-1)+1)\rceil)$

(2) d值确定:对上述时间复杂度关于d求极小值,最优解对应的d值即为所求解.

4.8

数值计算(时间t):

P_0: 28

P_5: (t0-1)-(L+2・o)=18, P_3: 18-g=14, P_2: 14-g=10, P_1: 10-g=6;

P_7=(t_5-1)- (L+2・o)=8, P_6=8-g=4;

P_4=(t_3-1)- (L+2・o)=4;

工作原理:

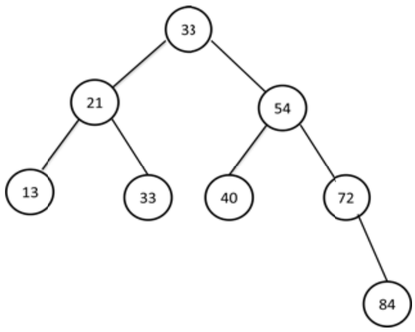
按时间顺序自底向上发送数据并求和,各节点在t-1时间完成局和的接收,然后在1个单位时间求和,求和完成后向其上一层发送计算结果,如此过程层层计算,对输入值求和.

4.10

1) 在第i(i≥1)步计算中,每一处理节点首先本地计算，然后将计算结果传输给编号为 $2^{i-1}+1$($2^{i-1}+1$≤最大编号)的处理节点;
2) 每一步计算之后,设置同步障;
3) 经log4(4个整数)步骤以后,每个处理节点的计算值即为所求前缀和.

第五章

5.3



存在多解,提供一种参考答案

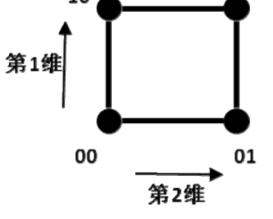
5.4

(1)原理:

把n个数根据处理器的数目分配，每个处理器有B=n/p个数，然后进行根据超立方体的维度d，进行1 到d 次循环，每次循环都把大于主元的数按维度发到编号比它大的节点上去，经过d 次循环后，大编号节点的数据都比小编号节点的数据大. 因此对各个节点做一次局部排序后，按节点编号大小连接，就成为一个有序的序列.

(2) 举例:

在2维超立方上执行串排序(33，21，13，54，82，33，40，72)



4 个处理器编号分别为：(00，01，10，11),d=log4=2

初始各个处理器分配的数：B(00)={33, 21}，B(01)={13, 54}, B(10)={82, 33}，B(11)={40, 72}
第一次循环i=1，选择主元x=33，按第1维方向发送和接收数据
处理器（00），B1={21,33}, B2=Φ，无发送数据
处理器（01），B1={13}, B2={54}，发送数据B2 给11
处理器（10），B1={33}, B2={82}，发送数据B1 给00
处理器（11），B1=Φ, B2={40,72}，无发送数据
第一次循环后，各个处理器：B(00)={21,33,33}，B(01)={13}，B(10)={82}，B(11)={40,72,54}

第二次循环i=2，选择主元x=33，按第2维方向发送和接收数据
处理器（00），B1={21,33,33}, B2=Φ，无发送数据
处理器（01），B1={13}, B2=Φ，发送数据B1 给00
处理器（10），B1=Φ, B2={82}，发送数据B2 给11
处理器（11），B1=Φ, B2={40,54,72}，无发送数据
第二次循环后，各个处理器：B(00)={21,33,33,13}，B(01)= Φ，B(10)=Φ，B(11)={40,54,72,82}
各个处理器局部排序后，再按编号连接成一个有序的序列。

5.7
P1:
WIT(1)=0, WIT(2)=1, WIT(3)=2, WIT(4)=4 非周期
P2:
WIT(1)=0, WIT(2)=1, WIT(3)=2, WIT(4)=5 非周期
P3:
WIT(1)=0, WIT(2)=1, WIT(3)=2, WIT(4)=5, WIT(5)=1, WIT(6)=2, WIT(7)=2 非周期

5.8
(1) WIT(1)=0, WIT(2)=1, WIT(3)=2, WIT(4)=4
(2)duel=p=6; P=6竞争获胜

第六章
6.3 (1)O$(logm)$
(2) m = 16 logm = 4
k(4) = m/logm = 4
A_0 = (0,1,2) A_1 = (7,9,11) A_2 = (16,17,18,19) A_3 = (23,24,25,27,28,30,33,34)
B_0 =(3,4,5,6) B_1=(8,10,12,13) B_2 = (14,15,20,21) B_3 = (22,26,29,31)
6.6(1) w(n) = O(n)
(2)初始：B(0,j) = j 1<= j <= 8
正向：B(1,1) = B(0,1)+B(0,2)=3
B(1,2) = B(0,3)+B(0,4)=7
B(1,3) = B(0,5)+B(0,6)=11
B(1,4) = B(0,7)+B(0,8)=15
B(2,1) = B(1,1)+B(1,2)=10
B(2,2) = B(1,3)+B(1,4)=26
B(3,1) = B(2,1)+B(2,2)=36
反向：C(3,1) = B(3,1) = 36

C(2,1) = B(2,1) = 10
C(2,2) = C(3,1) = 36
C(1,1) = B(1,1) = 3
C(1,2) = C(2,1) = 10
C(1,3) = C(2,1)+B(1,3) = 21
C(1,4) = C(2,2) = 36
C(0,1) = B(0,1) = 1
C(0,2) = C(1,1) = 3
C(0,3) = C(1,1)+B(0,3) = 6
C(0,4) = C(1,2) = 10
C(0,5) = C(1,2)+B(0,5) = 15
C(0,6) = C(1,3) = 21
C(0,7) = C(1,3)+B(0,7) = 28
C(0,8) = C(1,4) = 36

6.10 (1)t(n) = O(logn*logn)
p(n) = n

(2) D = (1,2,3,4,5,6,7,8)
第一轮：C = (8,6,3,6,7,2,2,1)
C = (8,6,3,6,7,2,2,1)
D = (8,6,3,6,7,2,2,1)
C = (1,2,3,2,2,2,2,1)
D = (1,2,3,2,2,2,2,1)
第二轮：C = (1,2,3,1,1,2,2,1)
C = (1,2,3,1,1,2,2,1)
D = (1,2,3,1,1,2,2,1)
第三轮：C = (1,1,3,1,1,1,1,1)
D = (1,1,3,1,1,1,1,1)

第七章

7.1 (1)划分：树的每一个节点都可以看做一个任务
(2)通信：若找到解，则返回根节点；若无解，则通知子节点进行计算
(3)组合：组合同一节点的子节点以减少通信次数
(4)映射：循环指派法

7.2 分治求和树
7.4 书 P170 图 7.9
计算次数 NlogN 通信次数 NlogN
7.5(1)同一列上 6 个网格点分配到同一个处理器
(2)设 0<= i <= 3，将第 i 列与第 i+4 列上的网格点分到同一个处理器
(3)分别按序分为：3*2 块，4*4 块。