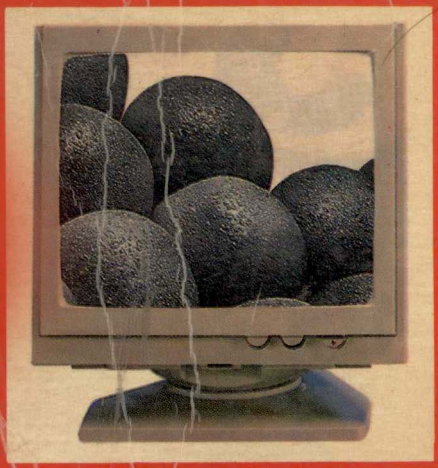
计算机语言技术系列丛书



并行程序设计方法

何炎祥 编著

学苑出版社

计算机语言技术系列丛书

并行程序设计方法

学苑出版社
1994

(京)新登字 151 号

内 容 摘 要

本书系统介绍了有关并行程序设计的基本原理和方法，着重讨论了如何利用信号量及 P,V 操作管程、会合和消息传递等方法解决并行程序设计中的同步、互斥、通讯、协作等核心问题，并以“生产者/消费者”、“读者/写者”、“有界缓冲区”、“顾客/服务员”、“哲学家用餐”等典型问题为例，详细讲叙了各种各样的并行程序设计方法并分析比较了它们的性能。各章之后附有练习以帮助读者提高并行程序设计的能力。本书层次清晰，例子解释详尽，易于理解，可供计算机专业的各类学生和教师作为教学用书，也可供从事有关研究和开发工作的广大科技人员参考。

欲购本书者，请与北京 8721 信箱联系，邮政编码 100080，电话2562329。

计算机语言技术系列丛书

并行程序设计方法

编　　著：何炎祥
审　　校：希　望
责任编辑：甄国宪
出版发行：学苑出版社　　**邮政编码**：100036
社　　址：北京市海淀区万寿路西街 11 号
印　　刷：建华印刷厂
开　　本：787×1092　1/32
印　　张：7.125　　**字　　数**：154 千字
印　　数：1～5000 册
版　　次：1994 年 4 月北京第 1 版第 1 次
ISBN7－5077－0776－8/TP・8
本册定价：8.00 元

序　言

本书简单讨论了并行程序设计的产生、发展及验证等问题，系统介绍了有关并行程序设计的基本原理和方法，着重讲叙了如何利用信号量及P，V操作、管程、会合、临界域和消息传递等方法解决并行程序设计中的同步、互斥、协作等核心问题，并以“生产者/消费者”、“读者/写者”、“有界缓冲区”、“顾客/服务员”及“哲学家用餐”等典型的并行程序问题为例，介绍了各种各样的解决方法。在叙述过程中，采用循序渐进、逐步深入的方式和形象生动的例子使得整个内容层次清晰，易于理解。

本书还特别注重各种方法在实际环境中的应用。书中除了分析解释一些能在实际系统上直接运行的并行程序例子外，还给了许多难度不一，有利于帮助读者提高并行程序设计能力的练习。为便于读者综合运用所述方法，我们还特辟专章介绍了一个实际的并发Pascal编译程序——Pascal－S编译器的设计思想及完整的源程序清单。本书可作为计算机专业的进修生、大学生、研究生及教师的教学用书，也可供从事有关研究和开发工作的广大科技人员阅读和参考。

非常感谢秦人华高级工程师为本书的出版所付出的辛勤劳动。

限于水平和经验，书中难免含有错误和不妥之处，恳请计算机界的同行及读者们不吝赐教。

何炎祥

一九九四年四月于武昌珞珈山

目 录

第一章 什么是并行程序设计 …… (1)

1.1 从顺序程序设计到并行程序设计 …… (1)

1.2 并行程序设计概念 …… (5)

1.3 并行程序的正确性 …… (8)

1.4 交错概念 …… (9)

1.5 操作系统与并行程序设计 …… (10)

1.6 本书概貌 …… (11)

1.7 本书中程序的表示方法 …… (13)

1.8 练习 …… (16)

第二章 并行程序设计的基础知识 …… (17)

2.1 进程的定义及特征 …… (18)

2.1.1 进程的定义 …… (18)

2.1.2 进程与程序的区别 …… (19)

2.2 临界段问题与互斥 …… (20)

2.3 进程间的同步和互斥关系 …… (25)

2.4 死锁 …… (26)

2.4.1 产生死锁的条件 …… (27)

2.4.2 解决死锁问题的方法 …… (27)

2.5 并行程序正确性的进一步讨论 …… (28)

2.6 操作原语 …… (29)

2.7 Pascal—S 中描述并行程序的方法 …… (30)

2.8 小结 …… (31)

2.9 练习 …… (32)

第三章　解决互斥问题的方法 …… (33)
3.1　引言…… (33)
3.2　最初的解决方法…… (34)
3.3　改进的解决方法…… (36)
3.4　第三种解决方法…… (38)
3.5　Dekker 算法 …… (40)
3.6　Peterson 算法 …… (41)
3.7　N 进程互斥问题的解决方法 …… (44)
3.8　利用硬指令解决互斥问题的方法…… (45)
3.9　练习…… (49)
第四章　信号量及其 P,V 操作 …… (54)
4.1　引言…… (54)
4.2　利用信号量解决互斥问题的方法…… (55)
4.3　“生产者/消费者”问题的解决方法 …… (58)
4.4　“有界缓冲区”问题的解决方法…… (62)
4.5　“嗜眠的理发师”问题的解决方法…… (64)
4.6　信号量及其 P,V 操作的实现 …… (68)
4.7　“选择性互斥”问题举例…… (70)
4.8　练习…… (72)
第五章　管程方法 …… (80)
5.1　管程的定义及性质…… (80)
5.2　用管程模拟信号量的方法…… (85)
5.3　用信号量模拟管程的方法…… (87)
5.4　“读者/写者”问题的解决方法 …… (92)
5.5　管程特性的证明…… (96)
5.6　练习…… (99)

第六章　Ada 语言中的“会合”机制 …… (104)
6.1　accept 语句及“会合”的基本特性 …… (104)
6.2　用“会合”模拟二元信号量 …… (108)
6.3　select 语句及其应用 …… (111)
6.4　“会合”特性的证明 …… (119)
6.5　练习 …… (120)
第七章　“哲学家用餐”问题及解决方法 …… (125)
7.1　“哲学家用餐”问题 …… (125)
7.2　第一种解决方法 …… (126)
7.3　第二种解决方法 …… (128)
7.4　一种正确的解决方法 …… (130)
7.5　条件临界域 …… (133)
7.6　条件临界域的实现 …… (136)
7.7　练习 …… (138)
第八章　消息传递方式 …… (140)
8.1　消息和消息传递 …… (140)
8.2　远程过程调用 …… (144)
8.2.1　过程调用与远程过程调用 …… (144)
8.2.2　RPC 的语义 …… (144)
第九章　并行程序设计语言 …… (148)
9.1　并发 Pascal …… (148)
9.2　CSP …… (150)
9.3　Ada …… (153)
9.4　并行程序设计语言模型 …… (156)
9.5　并行程序设计语言的特征 …… (158)
第十章　一个并发 Pascal 编译程序 …… (160)
10.1　概述 …… (160)

10.2 处理的对象…… (161)
10.3 P—code …… (163)
10.4 过程调用的处理…… (166)
10.5 并发性的实现…… (172)
10.6 信号量及其 P,V 操作 …… (175)
10.7 随机化…… (175)
10.8 完整的源程序清单…… (177)
参考文献…… (216)

第一章　什么是并行程序设计

1.1　从顺序程序设计到并行程序设计

图 1.1 是一个交换排序程序。此程序可由编译程序生成一组机器指令并在某个计算机上运行。该程序是顺序的，对于任何给定的输入数据(例如 40 个整数)该计算机将总是执行同一机器指令序列。

```
Program sortprogram;
  const n=40;
  var   a: array [1..n] of integer;
        k:integer;
  procedure sort (low, high:integer);
    var i,j, temp:integer;
    begin
      for i:=low to high-1 do
        for j:=i+1 to high do
          if a[j]<a[i] then
            begin
              temp:=a[j];
              a[j]:=a[i];
              a[i]:=temp
            end
    end;
  begin(*主程序*)
    for k:=1 to n do read (a[k]);
```

```
    sort (1,n);
    for k:=1 to n do write (a[k])
end.
```

图 1.1

还有一些比较好的顺序排序算法(参见 Aho 等人 1974 的著作),但这不在本书的讨论之列。在此,我们打算通过发掘这种排序中可以并行执行的部分来改善图 1.1 中算法的性能。例如,假定 n=10,输入数据为:4,2,7,6,1,8,5,0,3,9。将这组输入数据分成两半,即:4,2,7,6,1 和 8,5,0,3,9,再用两个伙伴程序同时分别对这两半数据进行排序,最后对已排好序的两半数据进行合并:

```
0
0,1
0,1,2
. . . . . .
```

通过简单分析该算法的复杂性就很容易看出,即使没有伙伴程序的帮助,该并行算法仍然比图 1.1 的顺序算法有效得多。在一个交换排序的内层循环中,有 $(n-1)+(n-2)+\ldots+1=n(n-1)/2$ 次比较,这近似于 $n^2/2$。但若只对 n/2 个元素进行排序,则仅需 $(n/2)^2/2=n^2/8$ 次比较。因此,这个并行算法只需 $2\times(n^2/8)=n^2/4$ 次比较就能将两半元素排好序,再用 n 次比较对这已排好序的两半元素进行合并即可。图 1.2 说明了在假定输入数据分别为 40,100,1000 个元素的情况下这个并行算法的优越性。

n	$n^2/2$	$(n^2/4)+n$
40	800	440
100	5000	2600
1000	500000	251000

图 1.2

图 1.3 是这个并行算法的顺序程序。该程序可以在具有 Pascal 编译程序的任何计算机上执行，它也很容易转换成其它的计算机语言程序。

```
program sortprogram;
  const n=20;
        twon=40;
  var a: array [1..twon] of integer;
      k: integer;
  procedure sort (low,high:integer);
      (*同图 1.1 的 sort 过程*);
  procedure merge (low, middle, high:integer);
    var count1, count2:integer;
        k, index1, index2: integer;
    begin
      count1:=low;
      count2:=middle;
      while count1<middle do
        if a [count1]<a [count2] then
          begin
            write (a [count1]);
            count1:=count1+1;
            if count1≥middle then
              for index2:=count2 to high do
```

```
                write (a[index2])
            end
          else
        begin
          write (a[count2]);
          count2:=count2+1;
          if count2 >high then
            begin
              for index1:=count1 to middle-1 do
                write (a[index1]);
              count1: =middle
            end
      end (* while 终止 *)
end;
begin (* 主程序 *)
  for k:=1 to twon do read (a[k]);
  sort(1,n);
  sort (n+1, twon);
  merge(1,n+1,twon)
end.
```

图 1.3

假定这个程序要在一个多处理机系统(含有一个以上 cpu 的计算机系统)上运行,那么,就需要某种表示法表达如何并行执行 sort(1,n)和 sort(n+1,twon)这个事实,在此,我们引入 **cobegin/coend** 这种表示法,如图·1.4 所示。假定 p_1, p_2,...,p_n 为过程名,那么 **cobegin** p_1,p_2,...,p_n **end** 意指:先暂时挂起主程序的执行,然后分别在不同的处理机上同时执行过程 p_1,p_2,...,p_n;当所有过程 p_1,p_2,...,p_n 的执行全部终止时,再重新继续主程序的执行。

```
program sortprogram;
  (*说明部分同图 1.3*);
  begin (*主程序*)
    for k:=1 to twon do read (a[k]);
    cobegin
      sort(1,n);
      sort(n+1,twon)
    coend;
    merge(1,n+1,twon)
  end.
```

图 1.4

除了 **cobegin/coend** 外，图 1.3 和图 1.4 中的程序是等同的。**cobegin/coend** 间的过程是否能并行地执行是由实现——系统硬件和软件决定的。向系统添加或从系统中去掉处理机不影响该程序的正确性，只会影响执行该程序的速度。

“concurrent”这个词是用来描述具有潜在并行执行能力的那些进程的。我们已经知道，一个算法可以通过找出其中可并行执行过程的方式来改善其性能，但只有真正实现了其并行执行时，才能获得最大的性能改善。

1.2 并行程序设计概念

并行程序设计是一种描述潜在并发性并解决与其相关的同步和通讯问题的程序设计表示法和技术。并发性的实现是计算机系统(硬件和软件)中独立于并行程序设计的一个课题。并行程序设计是很重要的，因为它提供了一种抽象的方法来研究并发性而不必顾及其实现细节。这种抽象性在编写清晰、正确的软件方面是十分有用的，这也是现代程序设计语言

所具备的基本特征。

编写并行程序的基本问题是如何识别出哪些部分是可以并行执行的。如果我们也将 merge 过程包括在 **cobegin/coend** 中(见图 1.5),那么,这个程序就不再是正确的了。例如,若将你合并数据的工作与两个伙伴进程的排序工作并行进行,那么,就可能出现图 1.6 中的情形。

```
cobegin
  sort(1,n);
  sort(n+1,twon);
  merge(1,n+1,twon)
coend
```

图 1.5

	伙伴 1	伙伴 2	你
开始	4,2,7,6,1	8,5,0,3,9	——
伙伴 1 交换	2,4,7,6,1	8,5,0,3,9	——
伙伴 2 交换	2,4,7,6,1	5,8,0,3,9	——
你合并	2,4,7,6,1	5,8,0,3,9	2
你合并	2,4,7,6,1	5,8,0,3,9	2,4
你合并	2,4,7,6,1	5,8,0,3,9	2,4,5

图 1.6

不过,如果有某种方法能将合并工作与分类工作同步进行,那么,merger 也可以看作是一个并发进程。

并发性不仅可以改善程序的性能,而且还可以提高程序的质量。例如,一个简单的批处理操作系统不应只看作是一个顺序程序,而应看作是可以并发执行的三个进程的有机体:

reader 进程、executer 进程和 printer 进程。其中，reader 进程从读卡机上读取卡片并将卡片上的信息存入输入缓冲区；executer 进程先从输入缓冲区中读出卡片信息，然后执行指定的任务，最后将执行的结果存入输出缓冲区；printer 进程则从输出缓冲区取出输出信息，并在行式打印机上打印出来（见图 1.7）。为完成一个给定的任务，这些并发执行的进程常常需要通讯和同步。进程间的通讯和同步一般有两种方式：经由共享变量（可供一个以上进程访问的变量）或利用消息传递方式。由于通讯和同步的作用，某个进程的执行就可能影响到其它进程的执行。例如，图 1.7 中的输入缓冲区就可看作是一个共享变量（在此称为共享缓冲区），它用于协调 reader 进程和 executer 进程间的工作。这两个进程的执行必须同步，例如，当输入缓冲区已满时，reader 进程就不得再向其中存入信息，而必须等待直至 executer 进程取走（一个）信息后才可继续工作。类似地，当输入缓冲区为空时，executer 进程决不可试图从中读取信息，而必须等待直至 reader 进程存入（一个）信息后才可继续工作。

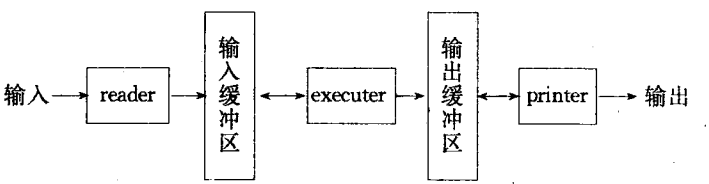


图 1.7　一个简单的批处理操作系统

1.3 并行程序的正确性

进行并行程序设计要比设计顺序程序困难得多，其难点主要表现在如何确保并行程序的正确性。

在考虑一个顺序程序是否正确时，通常采用测试法和断言法，所谓测试法即用许多组输入数据去执行该程序的每一分支，若对同一组输入数据去执行程序的每一分支，都能得到该程序的预期结果，那么就认为它是正确的。但正如 Dijkstra 所指出的：测试只能查看程序是否有错误，但不能保证程序的正确性。所谓断言法就是在程序的关键部位前后设置一些不变式。若在执行这些关键部位之前和之后这些不变式仍成立，则表明这些部位之间的程序(段)是正确的。然后将所有这些不变式合并起来就可构成该程序的一个逻辑证明。

以上方法对并行程序不见得有效。因为，第一，一个并行程序可能有许许多多不同的执行轨迹，以致无法穷尽。第二，组成一个并行程序的若干并发过程之间可能相互影响(或相互作用)这就不仅要求每个并发过程自身是正确的，而且还应保证它们不能因相互影响而"干扰"对方的正确性。第三，由于并发过程之间的相互影响(或相互作用)是利用通讯和同步方法进行的，因此，自然应考虑到实现通讯和同步的机制的正确性以及如何正确地使用它们。由此可见，在考虑并行程序的正确性时，我们不能简单地照搬在证明顺序程序正确性时所采用的那些方法，而需采用一些改进的或新的方法和技术。例如，前节的 sort 和 merger 过程中的基本正确性断言是：

sort 的输入断言：a 是一个整型数组；

sort 的输出断言:a 是“排好序”的,即 a 现在包含了初始输入元素的一个排列,且是按递增次序排列的;
merger 的输入断言:a 的两半部分是“排好序”的;
merger 的输出断言:a 的元素已按递增次序输出。

图 1.1 程序的正确性可直接根据过程 sort 的正确性获得。图 1.3 中程序的正确性可通过过程 sort 和 merger 正确性的“连结”来保证,讨论图 1.4 中程序的正确性时需要用到新的技术。例如,我们需要某种方法来描述过程 sort 的两个示例同时执行时彼此不会相互干扰。图 1.5 中的程序是不正确的,因为尽管它所包含的两个过程 sort 和 merger 本身都是正确的,但它们按所指方式并发执行后,由于其相互作用而致使该程序不正确(参见图 1.6)。要使它能正确运行,则需要使用同步原语。

1.4 交错概念

交错(interleaving)是使分析并行程序成为可能的一种技术。假定并行程序 P 由两个进程 p_1 和 p_2 组成,那么,我们说 P 运行的任何一个执行序列都可以通过交错这两个进程的执行序列而获得。我们认为这些执行序列穷尽了 P 的所有可能的行为。假定从 p_1 和 p_2 分别取出任意指令 I_1 和 I_2,如果 I_1 和 I_2 不存取同一存贮单元或寄存器,那么,无论是 I_1 在 I_2 之前执行,还是 I_2 在 I_1 之前执行,甚至是它们同时执行(若硬件允许的话),都无关紧要。此外,假定 I_1 是“将 1 存入单元 M”,I_2 是“将 2 存入单元 M”,如果 I_1 和 I_2 同时执行,那么,唯一合理的假定是其结果应是一致的。也就是说,单元 M 或存有 1 或存有 2,但决不可能既是 1 又是 2,也不可能是除此之外的某个