并行数据对象 并行处理系统 无等待同步 数据处理

计算机科学1996Vol.23 №.5

并行数据对象的无等待同步实现

陈华平 陈国良 杨 勃

（中国科学技术大学计算机系 合肥230027）

摘 要 A wait-free synchronization of a parallel data object is one important communication mechanism in shared-memory parallel processing system. A general method of describing parallel system by using I/O automata is introduced first, and then the concept of consensus number of a parallel data object and the consensus protocol of a wait-free synchronization are specified, and finally the consensus number of several commonly used data object and their proof are given.

关键词 Wait-free synchronization, Object implementation, Consensus number.

并行对象是并行处理系统中多个进程可共享的数据结构。执行这些并行对象的传统方法是利用临界区，即某一时刻只允许一个进程对该对象进行操作。然而，临界区不太适合于异步、容错系统[1-4]，如果一个出错进程滞留在临界区时，那么其它没出错的进程就不能往下继续运行。即使在一个不会出错的系统中，一个进程也可能遇到诸如缺页、高速缓存出错等故障而增加延迟时间。特别是在异构环境下的并行处理系统中，由于一些处理器比其它处理器的运算速度本来就快，或者有些存储单元内在的访问时间较长，所以上述问题也更为常见。

一个并行数据对象的"无等待执行"是指这样一种运行机制：任何进程在有限步内能够完成任一个操作，而与其它进程的执行速度无关[1]。无等待执行条件包含了容错性质，即不管其它进程是否发生意外的中止错误，或者执行速度任意改变，一个进程不可能被禁止完成一个操作。

一、并行系统的I/O自动机模型

在我们所讨论的问题中，计算模型由一组通过共享数据对象进行通讯的进程组成。每个对象有一个类型，该类型定义了一个可能状态集和一个原语操作集，只能通过这些原语才能对这数据对象进行操作。每个进程对数据对象实施一系列原语操作，主要包括激发操作和接收应答信息操作。

1.1 I/O自动机模型

为了对问题能进行形式化描述，我们利用I/O自动机的简化形式来定义计算模型[2]。一个I/O自动机A是具有下列部件的不确定自动机：(1)States(A)：一个状态的有限或无限集，其中包含一些确定的初始状态构成初始状态集Initial(A)。(2)In(A)：输入事件集。(3)Out(A)：输出事件集。(4)Int(A)：内部事件集。(5)Step(A)：由三元组(s',e,s)组成的转换关系集，其中s和s'是状态，e是事件。这样的一个三元组也叫作一"步"，它意味着处于状态s'的自动机可转换到另一状态s，并且这个转换与事件e有关。上述(2)、(3)和(4)三类事件可统一用Event(A)来表示，下面我们先给出一些定义。

定义1 如果$(s',e,s)\in$Step(A)，那么e在s'中是"允许的"(enabled)，否则就是"禁止的"(disabled)。

用$\Pi(s')$表示所有在状态s'中允许发生的事件集合，显然当且仅当$e\in\Pi(s')$时，e在s'中是"允许的"。I/O自动机必须满足这样的条件：输入事件不能是禁止的，也就是说，每一个输入事件e和状态s'，存在一个状态s和三元组(s',e,s)。

定义2 自动机A的一个"执行段"是状态和事件交替的有限序列$s_0,e_1,s_1,\cdots,e_n,s_n$或无限序列$s_0,e_1,s_1,\cdots$，其中$(s_i,e_{i+1},s_{i+1})\in$Step(A)。一次"执行"是$s_0\in$Initial(A)的一个"执行段"。一个"历史段"是在一个"执行段"中所出现的事件子序列，一段"历史"是一次"执行"中所出现的事件子序列。

如果一组自动机没有共享任何输出事件或内部事件，那么称这一组自动机相互之间是兼容的[2]。可通过一组兼容的自动机$\{A_1,A_2,\cdots,A_n\}$来构成一个新的复合自动机S，S的几个部件的具体构造方法如

陈华平 在职博士，现主要从事并行处理系统和并行程序设计环境的研究。陈国良 系主任，博士生导师，现主要从事并行分布计算和智能计算的研究。杨勃 博士生，现主要从事并行算法和并行计算环境的研究。

下：

(1)S 的状态集 States(S)＝{(a_1,a_2,…,a_n)|其中 a_i∈States(A_i)}，初始状态集 Initial(S)＝{(b_1,b_2,…,b_n)|其中 b_i∈Initial(A_i)}。

(2)S 的输出事件集 Out(S)＝Out(A_1)∪Out(A_2)∪…∪Out(A_n)。

(3)S 的内部事件集 Int(S)＝Int(A_1)∪Int(A_2)∪…∪Int(A_n)。

(4)S 的输入事件集 In(S)＝(In(A_1)∪In(A_2)∪…∪In(An))－Out(S)，即 S 的输入事件不能是某一自动机的输出事件。

(5)当且仅当对任一自动机部件 A_i 满足下列条件之一时，三元组(s'，e，s)才属于 S 的转换关系集 Steps(S)。①e 是 Ai 的事件，并且(s'，e，s)在 Ai 的投影属于 Step(Ai)，或者②e 不是 Ai 的事件，并且对自动机 Ai 的状态部件来说，s' 和 s 是完全相同的。

如果 H 是复合自动机 S 的一个历史，Ai 是复合自动机的一个组成部件，那么用 H|Ai 来表示由 Ai 的事件组成的 H 的一个子历史。

1.2 并行系统模型

一个进程 P 可看作是一个 I/O 自动机[1,2]，该自动机具有输出事件 INVOKE(P，op，X)和输入事件 RESPOND(P，res，X)，同时，INVOKE(P，op，X)也是对象 X 的输入事件，RESPOND(P，res，X)也是对象 X 的输出事件。这里 op 表示并行对象 X 的一个操作(可能包含一些变元值)，res 表示结果值，P 表示进程，X 表示对象，进程名和对象名都是唯一的，以确保进程和对象自动机是兼容的。

定义3 对于事件 INVOKE(P_1，op，X_1)和事件 RESPOND(P_2，res，X_2)，如果 P_1＝P_2，X_1＝X_2，那么这两个事件是匹配的。

如果一个进程历史以一个 INVOKE 事件开始，然后交替出现匹配的 INVOKE 事件和 RESPOND 事件，那么称这个进程历史是“良好的”。如果一个 INVOKE 事件后紧跟着的不是匹配的 RESPOND 事件，那么称该 INVOKE 事件是“未决的”(pending)。

一个并行系统{P_1，…，P_n；A_1，…，A_m}是由一组进程{P_i|i＝1，…，n}和一组对象{A_j|j＝1，…，m}构成的复合自动机，这里进程和对象通过对应的 INVOKE 和 RESPOND 事件复合在一起。设 H 是并行系统的一个历史，如果对于所有进程 P_i，H|P_i 是良好的，那么称 H 为良好的。如果并行系统的任一个历史都是良好的，那么称该并行系统也是良好的。下面我们只讨论良好的并行系统。

二、并行对象的实现模型

对象 A 的实现 I_A 是一个并行系统{F_1，…，F_n；R}，这里 R 也叫做对象表示，是实现 A 的数据结构；F_i 也叫做前端，由进程 P_i 调用来执行操作。具体实现结构如下图所示

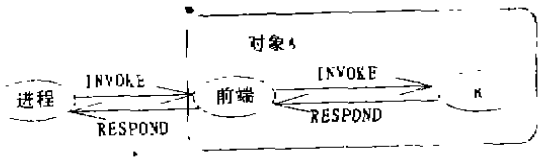

图1

(1)I_A 的外部事件也正好是对象 A 的外部事件；A 的输入事件 INVOKE(P_i，op，A)也是 F_i 的输入事件；A 的输出事件 RESPOND(P_i，res，A)也是 F_i 的输出事件。

(2)I_A 具有下列内部事件：R 的输入事件 INVOKE(F_i，op，R)与 F_i 的匹配输出事件相对应；R 的输出事件 RESPOND(F_i，res，R)与 F_i 的匹配输入事件相对应。

(3)为了简单起见，假定前端没有共享任何事件，它们通过 R 进行间接通讯。

用 I_i 表示 A_i 的一个实现，如果对于每一个系统{P_1，…，P_n；A_1，…，I_j，…，A_m}的每一个历史 H，存在一个{P_1，…，P_n；A_1，…，A_j，…，A_m}的历史 H'，H|{P_1，…，P_n}＝H'|{P_1，…，P_n}，那么 I_j 这个实现是正确的。

定义4 如果一个对象实现满足下列两个条件，那么称该对象实现是“无等待的”：①不存在这样的历史，其中经过 F_i 的无限步后，P_i 还有未决的 INVOKE 事件。②如果 P_i 在状态 s 中有一个未决的 INVOKE 事件，那么必定存在从 s 起始的一个历史段，该历史段全部由 F_i 和 R 的事件组成，并包含对该未决 INVOKE 事件的响应。

第一个条件排除了无限制的“忙等待”情况，即一个前端不可能经过无限步后还没对一个 INVOKE 事件进行响应；第二个条件排除了“条件等待”情况，即 F_i 不能阻塞等待其它进程使某条件为真。

三、几个常用对象的无等待实现问题

3.1 基本概念

在讨论一些常见并行对象的无等待实现之前，我们先给出“一致性”协议和“一致数”的概念。

一致性协议是 n 个进程通过一组共享对象{X_1，…，X_m}进行通讯的一个系统，每一进程从某一定义域内的一个输入值开始，它们通过对共享对象实施

一些操作来进行通讯，它们最终对某一公共值达成一致并结束。一般地，一致性协议必须满足下列三个性质[1,3,4]：

(1)一致性：不同的进程对某一公共值进行决定。

(2)无等待：经过一个常数步后每一进程决定返回值。

(3)有效性：公共决定值来源于某些进程的输入值。

定义5 并行对象X的一致数是能通过X达成进程间一致性协议，从而实现无等待同步的最大进程个数。如果这样的最大数不存在，那么称该并行对象的一致数是无限的。

3.2 读/写寄存器的一致数

读/写原子寄存器是并行系统常用的一个数据对象[5,6]，如果仅利用该对象进行通讯，那么即使是两个进程，也不存在该对象的无等待实现，也就是说，读/写原子寄存器的一致数为1。在给出该结果的证明之前，我们先引进几个述语。如果从当前状态往下执行会产生不同的决定值，那么称这个协议状态是"二阶的"，否则是"单阶的"。一个"决定步"是把协议状态开始由二阶变成一阶的一个操作。如果一个协议状态是单阶的，并且从该状态往下执行的最终决定值为x，那么称该协议状态是"x-值"态的。

定理1 读/写寄存器的一致数为1。

证明：(反证法) 假定通过原子读/写寄存器能实现进程P和Q的一致性协议。如果进程有不同的输入值，根据一致性协议的"有效性"条件，起始状态应该是二阶的。考虑从起始状态开始往下顺序执行：在第一阶段，P进程通过匹配的INVOKE和RESPOND事件执行一系列操作，直到到达这样的一个状态，从该状态往下的操作将使协议停留在一阶状态。由于进程P不能无限步地运行，并且不能阻塞，所以它最终肯定会到达这样的状态。在第二阶段，进程Q执行一系列操作也到达类似的状态。在往下的执行中，P和Q交替实施一些操作，直到执行一个决定步。由于协议不能永远运行，它最后肯定会到达一个二阶状态s，在该状态下任一个进程的操作是一个决定步。假定P进程的操作把协议引到"x-值"状态，Q进程的操作把协议引到"y-值"状态，这里x和y是不同的值。

(1)不失一般性，假定进程P的决定步是读一个共享寄存器(如图2所示)，s'是读操作后的状态。同时，协议有一个从状态s开始的历史段Hs，该历史段全部由Q的操作组成，并生成决定值y。既然状态s和s'的不同之处只在于P进程的内部状态，所以协议存在从s'开始的Hs与相同历史段，而这是不可能的，因为s'是"x-值"态，而s是"y-值"态。

(2)假定进程对不同的寄存器进行写操作(如图3所示)，按理P先写寄存器r，然后Q再写寄存器r'所产生的状态，与Q先写寄存器r'，然后P再写寄存器r所产生的状态应该相同，但一个是"x-值"态，另一个是"y-值"态，所以这是不可能的。

(3)假定进程对同一个寄存器进行写操作(如图4所示)，设s'是P执行写操作后的"x-值"状态，即存在从s'开始的全部由P进程的操作所组成的一个历史段Hs，该历史段最后生成值x；设s"是Q先进行写操作，然后P再执行写操作后所到达的"y-值"状态，由于P重写由Q先写进寄存器的值，s'和s"的不同之处只是Q的内部状态，所以协议存在从s"开始的与Hs相同的历史段，而s"是"y-值"状态，所以这是不可能的。 □

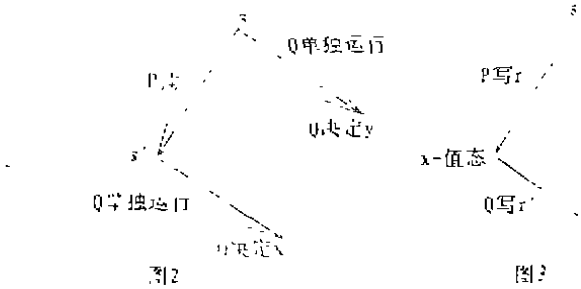

图2

图3

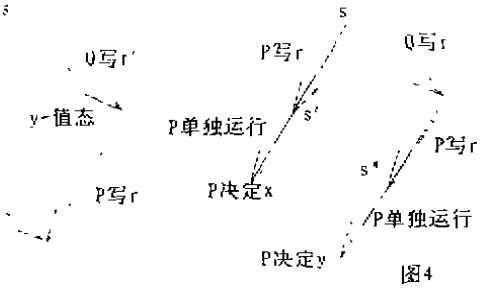

图4

3.3 队列、栈和表等数据对象的一致数

下面我们先考虑数据元素先进先出的队列结构，设enq(q)操作把一数据项放到队列q的末端，deq(q)操作从队列q的头部移出一数据项，如队列为空时执行deq操作，那么返回一个出错信息。下面定理2通过在队列上构造出一个二进程的一致性协议，从而证明队列的一致数至少为2；定理3用反证法说明在队列结构上不存在三进程的一致性协议，从而确定队列的一致数就为2。

定理2 队列的一致数至少为2。

证明：我们可以构造如下面图5所示的一个二进程一致性协议，为了简单起见，我们用decide(input value)return(value)表示抽象的对象操作，这样，协议的串行语义非常简单，即所有的对象操作返回第

一个 decide 的变元值，对队列进行如下的初始化设置，先放入元素0，然后放入元素1。每一进程从队列移出一元素，如果其值为0，那么返回它自己的输入值；如果其值为1，那么返回另一进程的输入值。

该协议满足一致性协议的条件，即①无等待：既然不包含循环，显然是无等待的。②一致性：如果每一进程返回自己的输入值，那么它们必须均移出元素0；如果每一进程返回另一进程的输入值，那么它们必须均移出元素1，而根据初始化设置这是不可能的。③有效性：假定移出元素为0的那个进程为胜者，那么在第一个队列操作之前，已对获胜进程的 prefer 位置进行设置，它也就是某进程的输入值。 □

```
decide(input,value)return(value)
  prefer[P]=input
  if deq(q)=0 then
    return prefer[P]
  else
    return prefer[Q]
  endif
end decide
```

图5

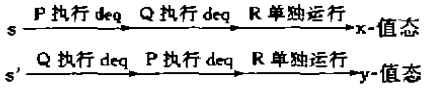

图6

定理3 队列的一致数为2。

证明：(反证法) 假定存在进程 P、Q 和 R 三个进程的一致性协议，类似定理1的证明方法，我们考虑 P 和 Q 将执行决定步时的状态，假定 P 进程的操作将把协议引入"x-值"状态，Q 进程的操作将把协议引入"y-值"状态，下面就各种情况分别讨论。

(1)假定 P 和 Q 都执行出队操作，如图6所示。设 s 是 P 进程先执行 deq(q)，然后 Q 进程再执行 deq(q)的协议状态；s' 是 Q 进程先执行 deq(q)，然后 P 进程再执行 deq(q)的协议状态。由于 s 是"x-值"状态的，所以存在以 s 起始的全部由 R 的操作组成的一个历史段 H_1，该历史段产生决定值 x。而 s 和 s' 的不同之处只在于 P 和 Q 的内部状态，所以协议应该存在以 s' 起始的与 H_1 相同的一个历史段，而 s' 是"y-值"状态，这是矛盾的。

(2)一个进程执行入队操作，另一个执行出队操作。不失一般性，假定 P 执行 enq(q)，Q 执行 deq(q)。由于 R 不能了解 P 和 Q 操作的先后次序，所以对 R 来说，P 和 Q 操作执行的先后顺序是可交换的，与(1)的情形相同，可以构造分别以 s 和 s' 起始的，全部由 R 的操作组成的两个相同历史段，而 s 和 s' 是不同值态的，这显然是矛盾的。

(3)假定 P 和 Q 都执行 enq 操作。考虑下面两种情况：设 s 是下列三个操作执行以后的状态：①进程 P 和 Q 按序分别把元素 a 和 b 入队。②运行进程 P，直到它把元素 a 移出队列。③运行进程 Q，直到它把元素 b 移出队列；设 s' 是下列三个操作执行以后的状态：①进程 Q 和 P 按序分别把元素 b 和 a 入队。②运行进程 P，直到它把元素 b 移出队列。③运行进程 Q，直到它把元素 a 移出队列。显然，s 是"x-值"状态，s' 是"y-值"状态。在进程 P 执行出队操作前，上述两种情况下 P 的所有操作都是一样的，由于在修改某些对象之前 P 已中止，所以，在进程 Q 执行出队操作前，上述两种情况下 Q 的所有操作也都是一样的，这一点对 R 来说是难于区分的，类似情形(1)，可以构造分别以 s 和 s' 起始的，全部由 R 的操作组成的两个相同历史段，而 s 和 s' 是不同值态的，这显然又是矛盾的。 □

对于栈、表和其它类似的数据对象，仿照定理2的方法可构造出类似于图5二个进程的一致性协议，并且仿照定理3的证明方法，也可证明在这些数据对象上不存在三个进程的一致性协议，所以它们的一致数为2。这些数据对象有一个共同之处，就是如果改变操作的次序，那么能返回不同的值。

结束语 本文主要讨论了并行数据对象的无等待实现问题，每个数据对象都有相应的一个一致数，该一致数代表了在该数据结构上能实现一致性协议的最大进程个数。一般说来，如果一个并行对象实现能确保某些进程在有限步内完成一个操作，而与进程执行的相对速度无关，那么称该并行对象实现是非阻塞的。非阻塞条件保证了整个系统不管个别进程的中断错误或延迟，整体上能够不断地正常进展。一个无等待实现必须是非阻塞的，但反过来并不成立，因为非阻塞实现允许有些进程空转(如循环测试)。

参考文献

[1] *Maurice Herlihy, Wait-free Synchronization*, ACM Trans. on Progam. Lang. Syst. Vol. 11, No. 1, January 1991

[2] N. A. Lynch 等, An introduction to input/output automata, Tach. Rep. MIT/LCS/TM-373, Mit Laboratory for Computer Science, Nov. 1988

[3] J. Aspnes 等, Fast randomized consensus using shared memory, J. Algorith. 11(1990)

[4] D. Dolev 等, On the minimal syschronism needed for distributed consensus, J. ACM 34(1)1987

[5] L. Lamport, Concurrent reading and writing, CACM *20(11), 1987*

[6] G. L. Peterson, Concurrent reading and writing, ACM Trans. on Program. Lang. Syst. 5(1)1983