计算机科学 1997Vol. 24№. 2

并行编程技术探讨

孙文辉 刘 平 曹东启
（中国科学院软件所 北京 100080）

摘 要 本文分别讨论了共享存储器处理机和多计算机结构这两种并行结构的并行编程方法和实现技术，这对于开发分布式编程环境，进行并行编程具有理论意义。

关键词 并行程序设计，共享存储器，处理机，多计算机。

1. 概述

随着并行机的发展和网络技术的成熟，如何将顺序程序设计成并行程序，如何设计分布式编程环境，以获得更大的加速比，这是90年代的重要研究课题。并行编程涉及面很广，而且并行系统硬件结构与并行算法的设计是紧密相关的。并行系统硬件结构主要包括两个方面，共享存储器处理机（图1）和多计算机（图2）。

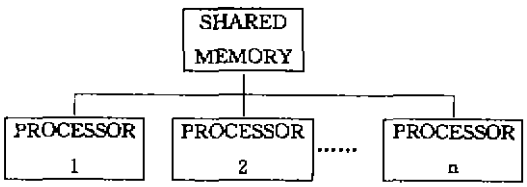

图1 共享存储器处理机结构

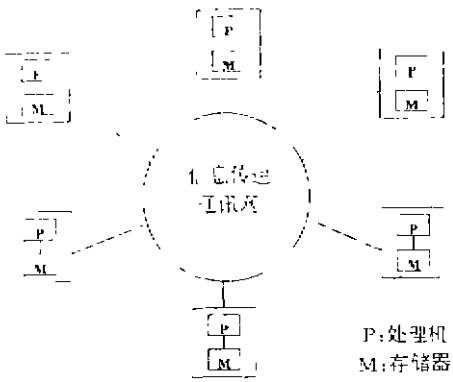

图2 多计算机结构

为了更有效地利用并行处理技术，顺序程序必须改造成有效的并行程序，一般所采用的技术如下：

1）数据并行：不同的数据并行地执行同一处理，此技术适合于数组和运算。

2）数据分区：这是一种特定类型的数据并行，数据空间被划分成相邻的区域，每个区域被不同处理机并行执行。

3）松散式算法：每个并行进程自己执行，进程之间相互没有同步或传输。

4）同步重复：每个处理机对不同部分的数据执行相同的重复计算，在每次重复之前处理机必须同步。

5）复制的工作者（Replicated worker）：维护一相同计算任务的中央池，有大量工作者进程从池中提取任务，执行所需的计算，并增加新任务到池中。当任务池空时，整个计算就终止。复制的工作者技术一般用于组合问题，诸如树或图搜索。

6）流水线计算：进程被安排在某种常规结构的各个结点上，诸如环或两维网。数据流过整个进程结构，每个进程执行整个计算的某一阶段。

在并行程序执行中，有下列问题可能限制并行程序的执行，降低加速比：

1）存储器竞争：当等待存取已被另一处理器使用的存储器时，处理器执行被延时，此问题仅发现在共享存储器的多处理机中。

2）过多的顺序代码，顺序代码的执行将严重限制能达到的最大加速比。

3）进程创建时间：并行进程创建时需花费一段时间。

孙文辉 博士生。刘平 深圳华为公司多媒体部总工。曹东启 研究员。

4)传输时延:此负荷仅发生在多计算机中,因为处理器通过信息传递通讯相互作用。

5)同步时延:当并行进程同步时,这意味着一个进程被迫等待另一进程。

6)负载不平衡:在一些并行程序中,计算任务以一种不可预料的方式产生,并当它产生时被赋给处理器。这可能导致有一些处理器空闲而另一些有较多的任务在执行。

2 共享式存储器处理机结构的编程

对于共享式存储器处理机结构而言,最简单的并行编程技术是松散式数据并行,并行进程之间不相互通讯或同步,这些程序包括如下类似结构创建大量并行进程,在每个元素执行相同计算:

```
FORALL I=1 To 100 Do
    A[i]=SQRT(A[i]);
```

FORALL 是 FOR 循环的并行形式,循环重复是并行执行的而不是串行的。类似于 FORK 操作符能将任一表达式变成与其它进程一起运行的一派生的并行进程,可以设置共享变量和私有变量,并行进程可以有共享数据和局部数据。进程也必须有足够大的粒度来克服与进程创建有关的负载。增加粒度的方法是让 FORALL 表达式中增加 GROUPING 原语来组合几个序号值到同一进程中执行。

为了增加存储器系统的能力,减少存储器竞争的机会,许多技术被用来改善处理机的性能,如高速总线,cache 存储器,多存储器模块和处理机-存储器互连网。在面向总线系统中,处理机通过高速公共总线连到单一的共享存储器,能有效地连到特定的总线的处理机的数目是受它的速度和共享存储器的速度限制的。一种有助于减小对共享存储器的存储速率的技术是高速缓冲存储器的使用。每个处理器有它自己的私有缓冲存储器来存储近来使用的存储器的备份。这样减小了通过公共总线来存取共享存储器的需要,但维护不同缓冲器中同一存储器位置的多个拷贝的一致性成为一个问题,一种解决缓冲器一致性的技术是在总线上广播所有写操作,以便它们能被所有缓冲器注意到。为了改善存储器系统的性能,它能被分成多个存储器模块,这允许许多处理器同时存取存储器,当然这假定它们引用不同存储器模块。为了在模块中更均匀地分布处理机存储器引用,一个地址"扩散"技术被用来将连续的存储器地址放置在不同存储器模块中。这种方法在存取大量数据数组,编写存储在存储器的代码时有助于减少竞争。有大量技术来连接处理机到多存储器模块,每个都具有自己的性能价格比。可通过 crossbar 互连网络获得最佳性能,但它的 $O(n^2)$ 价格是惊人的,也有许多通用的 O(nlogn) 互连网络,它包括蝴蝶网络和 shaffle 交换网络,这些网络能支持不同处理器的并行存储器请求。

大多数算法都是非松散式算法,需要进程相互使用。当进程开始修改将被其它并行进程使用,共享数据时,特定语言机制被需要来确保数据被目标进程读之前已完全被修改。通道变量这种新机制能存储一系列值来允许写进程在读进程之前进行。这些值以后能按需要一次一个反馈给读进程,若读进程较快,并试图读一通道,它将在一些其它进程写到通道之前被挂起。通道变量对于典型的生产者/消费者并行编程是理想的。通过使用通道以线性链方式连接生产者和消费者,创建一通道来进行并行处理。在通道算法中,一点数据值的流通过通道从一端传到另一端,每个进程执行对数据的特定的改变。

修改共享数据的需求将是确保在其它进程干涉之前修改被完成。通过锁 spinlock,完成其它进程所需的排斥,因而当企图锁同样的 spinlock 时驱使其它进程等待。在这一上下文中,原子操作被定义为不能被其它并行进程中断的数据更改。

来源于原子操作的并行进程的互斥创造了程序执行中的一顺序瓶颈。在共享数据对象高度使用的一些程序中,这导致竞争问题和性能下降。对竞争问题有许多特定类型的解决方法。可是一般规则是分布数据和它的存取控制机制。这种分散允许多个并行进程并行地存取共享数据的不同部分,因而降低竞争。总之通道(channel)用于进程通讯,spinlocks 用来对共享数据的原子操作。spinlock 仅适用于多处理机,通道仅适用于多计算机。

并行算法的主要一类是重复同步算法,大多数数据顺序数值算法和其它算法包括对大数据数组的一系列重复操作,通过赋予各进程不同部分的数据数组能达到并行。在某些情况下,进程能继续独立操作,达到很高加速比的松散式算法。可是许多算法要求在一重复过程中由给定进程产生的数据在下一次

重复过程中被其他进程使用。在这种算法中，进程必须在每次重复之后执行某类同步。同步的头一种方法是在每次重复之后让进程终止，然后在下次重复开始时重被创建。但在有很多进程创建负载时，这种进程终止方法是不可接受的。对于这些系统，一同步"barrier"能被使用来在每次重复过程之后同步进程，有使用 spinlocks 的 O(n) Barrier 技术，它是基于计数变量的，当通过 Barrier 的到达阶段时进程被计数，当最后一个进程通过到达阶段时，则打开随后的离开阶段以释放所有进程。为达到 O(logn)的 Barrier，使用一二叉树技术，当进程进入 Barrier 时找对来选择优胜者，优胜者在下次循环中再找对，这种"锦标赛"形式以 2 的倍数降低进程的数目。最后找到最终的取胜进程，取胜进程然后沿树下降，释放以前被阻塞的失败对手，最终所有进程被释放，返回计算。采用树方法中的分散策略增加了并行性，降低了共享通道的竞争。

3. 多计算机结构编程

处理器之间的通讯是通过连到每个处理机的硬件通讯接口实现的，物理传送、错误检测和寻址都需要时间，导致在计算中处理机之间发送数据的传输延时。各种不同的网络结构被设计来最小化每个处理机连接的链路数目，而尽可能最小地保证处理机之间的传输时延。有各种不同的计算机拓扑结构如线型、环型、2-D 网型、Torus、3-D 网型、Hypercuber。Hypercuber 拓扑结构能接受任何其他拓扑结构嵌入。这意味着执行在其它拓扑结构之一的任何并行程序将能很好地执行在 Hypercuber，换句话说，Hypercuber"包括"所有其他拓扑结构，加上为改善性能的其它附加连接。Hypercuber 嵌入是借助于二进制处理机数目的 Gray 码来实现，也就是说连续的数目仅在一个二进制位中是不同的。

多计算机的编程比起多处理机编程较复杂，由于耗时的多计算机之间的处理器之间传输时延，程序员必须识别机器结构，规划他们的程序来最小化传输负载量。简单地将共享存储器多处理机上的并行程序搬到多计算机结构中将会导致较差的性能，多计算机的本地存储器的分布特性要求程序以更分布形式组织，每个进程主要依靠它自己的本地变量。

并行编程的消息传递形式(message-passing)被设计来提供多计算机中执行的较高有效性，在消息传递形式中程序的组织反映了多计算机的基本组织。每个进程由一过程调用创建，并具有作为值参数传送的所需的共享数据。一旦创建，每个进程主要依靠它自己的局部变量，该局部变量存储在该处理机的本地存储器中，所有进程之间交互是通过消息传送的传输口实现的。通道变量被赋值来接受特定进程中的消息，创建具有两部分宣称的通讯端口是在主程序开始时的通讯变量名称和在进程创建表达式时的类似于 PORT 宣称。

@和%SELF 原语的使用将进程赋给处理机的特性已出现在几种并行编程语言中。其@特性允许程序克服缺省处理器分配策略，设置进程在任一物理处理机中，%SELF 表示自己的处理机号。通过意识到基本的多计算机拓扑结构和程序的进程通讯形式，程序能通过设置进程最优化程序性能。

写多计算机的并行程序在许多方面类似于多处理机编程，但主要不同的是多处理机能存储共享存储器中所有共享变量，这些共享变量能被所有处理机直接存取，而在多计算机中共享数据必须通过处理机的本地存贮器分布。处理机对它本地存储器中的存取是快的，但对远程存储器的数据存取需要耗时的消息传递。所以有效的多计算机编程要求数据在本地存储器中分区来最小化远程数据存取的需求。

最常用的例子是宇宙学的 N-天体问题，为了计算单一物体的力，所有其它物体必须被考虑，计算必须执行在每对物体中，最后该物体所承受的总的万有引力才能计算出来。在这程序中不管数据怎样分区，每个处理机仍需要存取所有的数据，尽管这个需求，它仍是可能地通过绕着环型通讯网循环数据的分区来创建一有效的多计算机程序，以便每个处理器最终接受每个分区的一个拷贝。通过部分重叠通讯和计算，将会最小化分区的循环所损失的时间。

这个虚环通讯结构可在环式多计算机拓扑结构中实施。通过嵌入虚环到高互连的拓扑结构，N 天体程序也能在其它拓扑结构诸如网状、Torus 或 Hypercube 上有效地实施，在前面所说 Gray 码排序方案能被使用来嵌入线、环和网的虚拟拓扑，成为一个物理的 Hypercube 拓扑结构。

第二个例子是矩阵乘法 A×B，这一例子中大多

数问题类似于N天体问题，但是以二维代替一维。矩阵乘法的数组在Torus多处理机中两维处理机器的设置下被分区成两维。对于这一分区，每个处理机需要Torus它自己行中所有A分区的拷贝和B分区中它自己列的所有B分区的拷贝。为了实现该需求，所有A分区通过它们的行旋转，所有B分区通过它所有列旋转。由于Torus是端相互连接的，它是分区口两维旋转的理想拓扑结构，假如分区的大小大得足以产生有效的进程粒度来克服进程创建和开始的数据分布负载的话，矩阵乘法程序也能达到较好的加速比。体现在语言中，所有的进程是通过调用过程来创建的，它主要是依靠它自己的本地变量。在程序的计算阶段，数据是通过使用通讯口在进程之间通讯的。

复制的工作者是重要的并行编程技术。这一技术适用于当程序执行时计算活动以一种不可预料的方式产生。对此情况，简单的嵌套FORALL循环是不行的。当程序动态产生时必须动态组织程序来存储和分布很大数量的计算任务。工作池的概念是一个重要概念，每个计算所需的单元由存储在工作池中的任务描述符表示，一组相同的工作进程连续地从工作池中读和写任务描述符。当工作者进程空闲时，它从工作池中读任务描述符，执行所需的计算。在该计算中，新的任务描述符由工作者产生，并被加入到工作池中。工作池的实施也产生竞争，为避免通道竞争，工作池被分布成较小数目的通道，每个通道有本地组的工作进程，这一分布引入了当某些工作组空闲而另一些工作组有许多工作项需处理时负载平衡的问题。通过工作项的简单分配技术来克服这一问题，诸如每个工作者在工作池中的所有通道中使用round-robin分布。

终止检测也是复制的工作者程序中一个有趣论点，当工作池为空时，程序将终止所有工作者进程都在等待更多的工作，当工作池变为分布的时候，这一条件更难以检测。对于分布式多处理机，可以使用一个计数数组来检测终止，记录等待在那个通道的空闲工作者的数目。有许多并行算法使用了复制的工作者技术，最简单的例子是N皇后。

分布式工作者也是多计算机中的并行技术，在这里工作项被一套相同的工作者进程动态创建和交换。这些程序包括Getwork过程来取回工作项，Putwork过程分配工作项到其它工作者中。工作者过程随着特定的应用不同而不同。然而Getwork和Putwork过程则是基本不变的，有一些主要分布式工作者程序决定Getwork和Putwork的特定执行。Putwork实施取决于是否工作者是不可分的还是可分的。若工作者是不可分的，那么任何一个工作项可被送到任一工作者，所以Putwork过程是自由地分配所创造的工作项以平衡工作者之间的负荷。

Getwork过程不依靠工作者是否是不可分的，Getwork的实施的重要因素是终止检测的方法和是否使用工作压缩。若被假定在特定机器上信息传输有一个已知的上界，那么有几个简单的终止检测方法。若使终止检测是更可靠的，必须考虑信息的任意的传输时延问题，解决该问题通过使用信息确认，由Getwork发往每个已接受的工作项的源。Getwork必须保持所有未确认信息的数目，将在所有确认接收到之后才允许终止。可通过活动着的工作者树检测到终止，其终止条件是：所有工作者正等待工作，并且没有工作项在工作者之间传播。

结束语　本文分别讨论了共享存储器处理机结构和多计算机结构的并行编程方法和实现技术难点，目前有许多较成熟的一些分布并行编程环境，这当中有PVM、C-Linda、SR、FortranM等，相信通过我们一段时间的努力，分析和消化这些系统，是完全有能力设计出自己的并行编程环境及并行语言的。

（参考文献略）