计算机科学 1996 Vol. 23 №. 6

# 分布存储环境下 SPMD 执行模式的任务调度问题

陈华平　周学海　陈国良

（中国科学技术大学计算机系　合肥 230027）

**摘　要**　The task scheduling model in parallel distributed computing and communication model of SPMD schema are introduced first in this paper. Then it is mainly disscussed how to use the closed-form formulas to determine the optimal number of processors which execute data parallel programs with different computing complexity according to the master/slave or tree-structured form. Lastly, the influence of communication delays on performance of parallel programs is analysed in distributed memory environments.

**关键词**　Task scheduling, SPMD executing schema, Communication overhead.

在并行分布计算中，用任务图来表示并行程序中各任务之间的执行关系，然后对任务图进行有效调度是提高并行程序执行性能的一种有效方法。当任务图中包含比较有规律的子任务图时，如任务图中包含有在不同数据上执行相同例程的多个子任务，并且这些子任务之间具有主从方式或树形方式的 SPMD 执行模式时，我们可把这些规律性较强、结构比较固定的多个 SPMD 任务构成的子任务图作为整个任务图的部件，并把这些部件通过分层组织结构构成复合任务图[1]。这样，复合任务图的结点可能是一个简单任务，也可能是从层次上可以进一步分解的子任务图。

在分布存储的并行计算模型上，通讯延迟的存在使任务调度更为复杂，它必须在尽可能利用各任务之间的并行性和尽量减少通讯延迟之间进行折衷[2,3]。如果无视通讯延迟的存在，而只考虑尽量增大并行执行时间，那么有可能在多个处理器上的执行效率还不如在一个处理器上的串行执行效率。同样，如果没有完全开发任务之间的并行性，那么并行处理器就没得到充分使用。一般情况下，任务调度目标就是要获得最小的调度长度，即整个并行程序的执行时间。

对复合任务图的一种调度办法是先对各子任务图进行调度，由于子任务图一般比较有规律，所以能用闭式表达式来决定每个子任务图应使用的最佳处理器数，然后把各分调度结果合并成一个统一的调度方案[2]。在本文中我们主要讨论了如何静态地把以 SPMD 模式执行的各子任务按最佳处理器数分配调度到分布存储的并行计算模型上，以及如何根据各子任务的计算复杂度和一些通讯延迟参数来决定任务调度所使用的最佳处理器数目，最后讨论了常用的几种复合任务图的调度办法。

## 一、模型和定义

### 1　任务调度模型

一般地，可用(PT, Sche, TM)来表示调度模型，其中 PT 表示并行程序任务，TM 为目标机器，Sche 为调度方案。

一个并行程序可用(A, ≺, $[D_{ij}]$, INS($a_i$))来刻画，其中 $A=\{a_k \mid k=1,2,\cdots,n\}$ 为任务集，"≺"定义了任务间的部分有序关系。"$a_i \prec a_j$"表示任务 $a_j$ 的执行依赖于任务 $a_i$ 的执行，也就是说，任务 $a_i$ 必须在任务 $a_j$ 之前执行。$[D_{ij}]=\{D_{ij} \mid i,j=1,2,\cdots,n\}$ 是一个 $n \times n$ 的通讯矩阵，$D_{ij}(\geq 0)$ 表示任务 $a_i$ 传送给任务 $a_j$ 的数据量。INS($a_i$)是个函数，表示任务 $a_i$ 的工作负载，有时就表示该任务的执行时间。在并行分布计算中，PT 也可用任务图 $G=(V,E)$ 来表示，其中 $V=\{a_i \mid i=1,2,\cdots,n\}$ 为表示 n 个任务的有权结点，有时

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。周学海　在职博士，现主要从事并行图象处理的研究。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。

也可直接用 i 作为任务 $a_i$ 在 G 中所对应的结点标识，任务 $a_i$ 的工作负载 $INS(a_i)$ 为该结点的权重；$E=\{(a_i,a_j)|a_i,a_j\in V\}$ 为表示任务间通讯关系的带权有向边，它表示任务 $a_i$ 必须在任务 $a_j$ 之前执行，任务 $a_i$ 与 $a_j$ 之间的通讯量 $D_{ij}$ 为该有向边的权。

一般地，目标机器假定由 m 个处理单元组成，该 m 个处理单元可以是同型的，也可以是异型的，它们通过任意的互连网络进行连接。每个处理器某一时刻只能处理一个任务，每个任务可在任何一个处理器上运行。可用六元组 $TM=(P,[P_0],[S_i],[I_i],[B_i],[R_{ij}])$ 来刻画目标机器的特性，其中：(1)$P=\{P_1,P_2,\cdots,P_m\}$ 为构成并行结构的一组处理器；(2)$[P_0]$ 为 $m\times m$ 的互连网络拓扑矩阵；(3)$S_i(1\leqslant i\leqslant m)$ 为处理器 $P_i$ 的执行速度；(4)$I_i(1\leqslant i\leqslant m)$ 表示在处理器 $P_i$ 上启动一个消息传递所需的时间；(5)$B_i(1\leqslant i\leqslant m)$ 表示在处理器 $P_i$ 上启动一个进程执行所需的时间；(6)$R_{ij}(1\leqslant i\leqslant m,1\leqslant j\leqslant m)$ 为两个相邻处理器之间的消息传送速率。

并行程序任务的一个调度 Sche 其实就是任务图 G 到目标机器的一个映射 $f:A\mapsto\{1,2,\cdots,m\}\times[0,\infty)$，$f(a_i)=(p,t)$ 表示任务 $a_i$ 被调度到编号为 p 的处理器上，起始执行时间为 t。一般地，我们可用 Gantt 图 $GC=\{(P(a_i),T(a_i))|a_i\in A\}$ 来直观地表示调度结果，其中函数 $P(a_i)$ 表示分配给任务 $a_i$ 的处理器号，$T(a_i)$ 表示任务 $a_i$ 的起始执行时刻。

## 2 通讯模型

在下面的讨论中，为了简单起见，我们有以下假定：(1)$P_{ij}=1$，即处理器是全连接；(2)每个处理单元的执行速度相同，均为 α，那么任务 $a_i$ 的执行时间可用任务的工作负载 $INS(a_i)$ 来代表；(3)每两个相邻处理器之间的消息传输率相同，为常数 $R_{ij}$，那么 $b=1/R_{ij}$ 表示每两个相邻处理器之间传送单位数据量所需要的时间。(4)每个处理器的消息传递启动时间相同，均为 a。在大多数情况下进程启动时间与进程执行时间相比可忽略，所以假定 $B_i$ 为零。

在以上定义的基础上，我们可把通讯函数定义为 $u(x)=a+bx$，其中 a 为消息传递启动时间，b 为传送单位数据量所需要的时间，x 为通讯数据量大小。并且假定消息发送者每次只能发送一个消息，而消息接收者可缓冲接收所有传来的消息。

## 二、主从方式下的数据并行任务调度

在 SPMD 模式的并行分布计算中，经常使用如图 1 所示的主从式程序执行方式，它的主要思想是由一个主处理器（不妨设为 $P_0$）负责接收需要处理的输入数据，然后把输入数据 S 划分成等量的 k 份后组成 k 个子任务 $a_1,a_2,\cdots,a_k$，分别传送给 k 个从处理器 $P_1,P_2,\cdots,P_k$。设这 k 个从处理器执行相同的任务例程 F，即所有子任务具有相同的执行时间复杂度函数 F(x)（其中 x=S/k），它也代表了这些子任务的工作负载 $INS(a_i)(i=1,2,\cdots,k)$。每个子任务计算完后分别把各自的结果 R 送回给主处理器，由主处理器 $P_0$ 来完成最后的合并操作。例如求一串数的和、平均、最大值、最小值及并行搜索数组等都是这一类的问题。

我们假定 $P_0$ 在一个常数时间 g 内完成划分预处理操作，在常数时间 h 内执行完合并操作，并且任务 F 的返回值数据量 $R\ll S$。$P_0$ 每发送一个消息所需时间为 a+bx，所以收到最后一个消息的处理器 $P_k$ 必须在 g+k(a+bx) 时才能开始执行（已把 β 忽略，下同），其中其它在此之前接收到消息的处理器已开始并行执行任务 F。$P_k$ 执行 F(x) 段时间后，经过 a+bR 时间把结果传送给任务 H 的缓冲器，最后 $P_0$ 经过 h 时间完成合并操作。根据上面所讨论的通讯模型可知，第 i 个子任务 $a_i$ 所分配的处理器号为 $P(a_i)=P_i$，起始执行时刻为 $T(a_i)=g+(a+bS/k)*i$，整

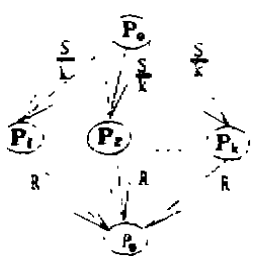


图1 主从式执行模式

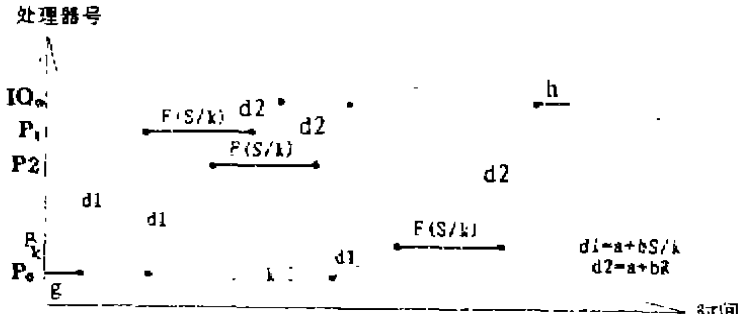


图2 主从方式下各子任务的执行时序

个调度结果如图2的执行时序图所示。其中图中虚线表示通讯延迟，d1为每次分送数据的通讯开销，d2为结果返回给$P_0$的通讯开销。由于$P_0$可以缓冲接收消息，所以接收子任务结果可重叠进行，为了清楚起见，在图中用$IO_0$表示$P_0$重叠接收结果的缓冲器。

设T(S,k)为整个执行时间，那么

$$T(S,k)=g+k(a+bx)+f(x)+(a+bR)+h$$
$$=C+ak+f(x) \quad (1)$$

其中$C=g+h+a+bR+bS$，$x=S/k$。下面的问题就是要选择一个介于1和min(S,m)之间的k值，使得T(S,k)取得最小值。这本身就是一个常见的定量并行粒度大小问题，就是如何平衡计算时间和通讯时间。如果一个处理器上分配太多的任务，那么由于失去部分并行性而会降低效率，如果分配太少的任务，那么由于通讯开销也会使效率下降。

为了求使得T(S,k)值最小的k值，可对(1)式中的k求微商，设使微商等于0的值为$k^*$，即：

$$a+\frac{\partial F}{\partial k}=0;k^*\in[1..\min(S,m)] \quad (2)$$

下面我们就各子任务F具有不同时间复杂度的情况下分别进行讨论。

假设F(x)为多项式复杂度，不妨设$F(x)=x^n=(S/k)^n$，$\frac{\partial F}{\partial k}=-\frac{nS^n}{k^{n+1}}$，由(2)式可得：

$$k^*=\min(\sqrt[n+1]{\frac{nS^n}{a}},m) \quad (3)$$

这个结果的含义也是非常明确的，它说明了：(1)通讯启动开销a对取得最优并行性能有直接影响。在通讯开销较大的计算环境下，所使用的最佳处理器数目就要少一些，相应的数据并行计算粒度就会大一些；(2)在计算数据量S增大的情况下，应选择多一点的处理器来并行执行这些数据；(3)当每个子任务的计算时间复杂度增大时，从并行中也就能获得更大的收益。这个结果也是非常直观的，就是当每个处理器对等量数据的计算复杂度增大时，应尽量减少通讯所带来的额外开销。

式(3)这个结果也说明了为什么在具有较高通讯开销的分布存储并行计算系统中，有时很难取得令人满意的加速比。不妨假定F(x)=x，并且使用最佳处理器数来并行计算，这时所得到的加速比为

$$S_p=\frac{F(S)}{T(S,k^*)}=\frac{F(S)}{C+ak^*+f(S/k^*)}$$
$$=\frac{F(S)}{g+h+a+bR+bS+2\sqrt{aS}}$$

当上式中S比其它任何量都大很多时，$S_p\approx O(S)/(bS+2\sqrt{aS})$。这说明消息启动开销a和消息传递速率b对$S_p$的影响较大，尤其是消息传递速率b成为影响$S_p$的主要因素。假定a=0，那么当b=1，即单位数据的通讯成本与执行成本相同时，就有$S_p\leqslant 1$，这个结论也是非常明显的，就是说对于线性时间复杂度的程序来说，如果单位数据的通讯成本与执行成本相同，那么在多个处理器上的并行执行效率还比不上在单个处理器上串行执行该程序的效率。

## 三、树形方式下的数据并行任务调度

树形方式下的SMPD并行执行模式是从某个处理器开始(不妨设为$P_0$)，不断地向其它邻接处理器分发数据。每个处理器自身处理一部分输入数据，并把剩余输入数据按一定比例、按一定次序往其它处理器传送。一般地，假定有$m=2^h$个处理器，那么可把数据量S分成大小为$S/2^h$的m份，每个处理器按一定规律把需要传送的数据分发到其它处理器上。如果处理器接收到的数据量已小于某个值，那么就可以不再分发给其它空闲处理器。每个处理器执行完后，把结果返回给传送原始数据给它的处理器，由该处理器再向上传递，显然，在树形SPMD模式下，我们并不一定需要上面第2节中的全连接假定$P_0=1$。下面图3是k=3时的树形执行方式，图4是表示各子任务的执行时序图。

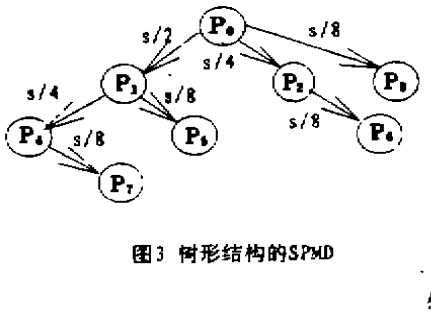


图3 树形结构的SPMD

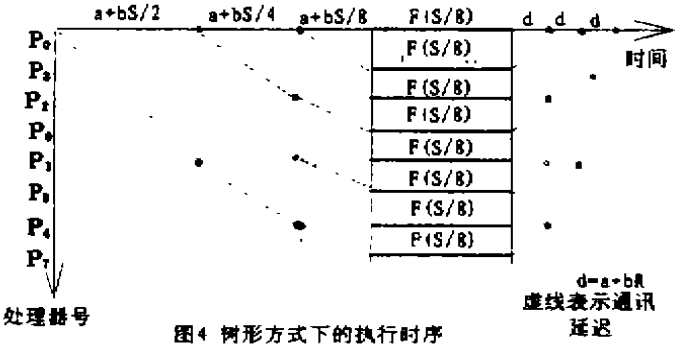


图4 树形方式下的执行时序

注意上面树中分发数据的次序是严格按照从左到右的，一般情况下，第i层分发数据的通讯开销为

$a+b\cdot(S/2^i)$，所以分到第 k 层结点的整个通讯时间为 $\sum_{i=1}^{k}(a+b\frac{S}{2^i})$，结果从最低层传到根结点的时间为 k(a+bR)。从图 4 可看出，按图 3 的树形 SPMD 方式执行时，所有子任务同时开始执行，起始执行时刻为 $\sum_{i=1}^{k}(a+b\frac{S}{2^i})$，每个任务的执行时间为 $F(S/2^k)$，所以整个并行程序的执行时间为

$$T(S,k)=\sum_{i=1}^{k}(a+b\frac{S}{2^i})+k(a+bR)+F(S/2^k)$$

化简后可得

$$T(S,k)=bS(1-1/2^k)+k(2a+bR)+F(S/2^k) \quad (4)$$

当 k=0 时，该数据并行算法就演变为在一个处理器上执行的串行算法。现在我们就是要找一个 $k^*$，使得 T(S,k)最小。下面我们就不同时间复杂度的 F(x)来加以分析。理论上说来，如果不考虑通讯成本的话，当 k=LogS 时将获得最大加速比[2]，当 S 比较大时，$1-1/S\approx 1$，所以其加速比为：

$$S_p=\frac{T(S,0)}{T(S,LogS)}=\frac{F(S)}{F(1)+(2a+bR)LogS+bS(1-1/S)}\approx\frac{F(S)}{F(1)+(2a+bR)LogS+bS}$$

很明显，通讯参数 a 和 b 对加速比 $S_p$ 有直接影响，当 S 较大时，$S\gg LogS$，这时消息传递速率 b 对加速比的影响更大一些。

我们假定各子任务的执行时间复杂度是线性的，即 F(x)=x，这时 $F(S/2^k)=S/2^k$，所以

$T(S,k)=(S/2^k)(1-b)+k(2a+bR)+bS$，令

$$\frac{\partial T}{\partial k}=\frac{(b-1)(\ln 2)S}{2^k}+(2a+bR)=0 \quad (5)$$

显然当 b>1 时式(5)无解，这时取 $k^*=0$，即只使用一个处理器。这表明当通讯成本较高时，用树形 SPMD 结构方式来执行一个线性复杂度的算法时，它的并行执行效率还比不上串行执行性能。当 b<1 时，$k^*=Log(\frac{(1-b)(\ln 2)S}{2a+bR})$，显然当 a 和 b 增大时，$k^*$ 减少，这表明通讯开销增大时，应减少执行处理器的数目，相应地增加了并行计算粒度。当 a 和 b 一定时，随着 S 的增大，$k^*$ 也增大，这个结果也是非常直观的，即工作负载增大时，应使用多一些的处理器来并行处理这些数据。

如果 $F(x)=x^2$，这时 $F(S/2^k)=(S/2^k)^2$，所以

$T(S,k)=(S/2^k)^2-b(S/2^k)+k(2a+bR)+bS$，令

$$\frac{\partial T}{\partial k}=\frac{bS(\ln 2)}{2^k}-\frac{2S^2(\ln 2)}{2^{2k}}+(2a+bR)=0 \quad (6)$$

这时 $k^*=Log(S)-Log(Y)$，其中，$Y=\frac{b}{4}+(\frac{1}{2})\sqrt{\frac{b^2}{4}+5.771a+2.885b}$。例如，当 a=250，b=10，R=10，S=1000 时，Y=23，$k^*\approx 5$，而不考虑通讯的情况下 $k^*=Log(1000)\approx 10$，所以在考虑通讯开销的情况下，为了获得最大的加速比，所使用的最佳处理器数目要比估计值少。

**结束语** 本文主要讨论了分布存储环境下 SPMD 并行执行模式的任务调度问题，一般说来，对这些有规律的子任务图可用闭式表达式来决定所使用的最佳处理器数。由若干个子任务图按一定层次可构成复合任务图，对复合任务图最简单的一种调度方案是采用"自顶向下"的方法[1]，即按前面所讨论的时间估计办法先调度上层子任务图，然后依次独立地调度下层子任务图，这种方法实现起来简单，但有时不很有效，因为它忽略了存在于不同子任务图之间的并行性。另一种调度办法是先通过深度优先遍历算法把分层结构的任务图转换成一般任务图[2]，然后采用一些启发信息来进行调度，如基于任务归类的"聚类"算法[4]，或基于按任务优先级排序的就绪任务队列结构的调度算法[5]等。

## 参考文献


[1]T. G. Lewis, Foundations of parallel programming, TR, Oregon State University, Computer Science Dept., 1993

[2]Hesham El-Rewini et al., Task scheduling, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994, 07632

[3]A. F. Bashir et al., A static study of a task scheduling algorithm, IEEE Trans. Comput., C-32, no. 8, 1983

[4]H. El-Rewini and T. Lewis, Scheduling paralleling program tasks onto arbitrary target machines, J. Parallel and Distributed Computing, 1990

[5]A. F. Bashir et al., A static study of a task scheduling algorithm, Same to [3]