计算机科学1997 Vol. 24 №. 1

并行分布计算中的任务调度问题(一)

陈华平 李 京 陈国良
(中国科学技术大学计算机系 合肥 230027)

摘 要 Task scheduling is one of the most common and challengable problems in parallel and distributed computing and it has great influence on the performance of parallel programs. This paper has introduced the basic concept and model of task scheduling in parallel and distributed computing first, then surveyed the optimal and heuristic algorithms in static scheduling.

关键词 Parallel computing, Distributed computing, Task scheduling, Surveys

1 引言

任务调度是并行分布计算中最具有挑战性的问题之一,其一般形式和几种受限形式都是NP完全问题[1][2]。要取得多项式时间复杂度的最优调度算法,就必须对表示并行程序的任务图或并行计算模型附加一些限制条件,但在已进入大规模并行处理的今天,并行分布计算模型和表示并行程序的任务图是多种多样的,因此,如何针对不同的具体并行程序任务而采用效的调度方法,已成为影响并行程序执行性能的一个主要因素[3]。本文主要概述了并行分布计算中经常使用的一些任务调度方法,并分析了它们的主要算法思想和各自存在的优缺点。

一般说来,并行分布计算中的任务调度方法主要可分为静态调度、动态调度和混合调度这三种技术[4]。静态调度是在并行程序编译时就决定每个任务的执行处理器及执行时刻,经常用于任务图比较确定的情况下。动态调度则是在并行程序运行过程中根据当前任务调度情况,临时决定每个任务的执行处理器。动态调度主要用于任务图不确定的情况下,但它不可避免地会带来额外开销[5]。混合调度是介于静态调度和动态调度两者之间的调度方法,在编译时静态调度部分任务,而剩余部分则采用动态调度方法。

在静态调度中,如果一个调度算法能在多项式时间内获得最佳调度,那么称之为有效的最优调度算法。但是,只有在极少数情况下才存在有效的最优调度算法,大部分最优调度所需的执行时间随任务数或处理器数目呈指数增长。因此,经常采用启发式任务调度方法[6][7]来静态地把各任务调度分配到各处理器上,它虽然不能确保获得最优解,但可以获得最优调度的近似解。

2 任务调度的基本模型

2.1 任务图

一个并行程序的性质可用$(A,\prec,[D_{ij}],W_i)$来刻画,其中$A=\{a_k|k=1,2,\cdots,n\}$为任务集,$[D_{ij}]=\{D_{ij}|i,j=1,2,\cdots,n\}$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;$W_i$表示任务$a_i$的工作负载;"$\prec$"定义了任务间的偏序关系,"$a_i\prec a_j$"表示任务$a_j$的执行依赖于任务$a_i$的执行。在并行分布计算中,偏序关系"$\prec$"一般用任务图$G=(V,E)$来表示,其中$V=\{1,2,\cdots,n\}$为表示n个任务的有权结点,任务$a_i$的工作负载$W_i$为该结点的权重;$E=\{(i,j)|i,j\in V\}$为表示任务间通讯关系的带权有向边集,任务$a_i$与$a_j$之间的通讯量$D_{ij}$为这些有向边的权。$IMP(i)=\{k|(k,i)\in E\}$表示任务图中结点i的直接前趋势,$IMS(j)=\{k|(j,k)\in E\}$表示任务图中结点j的直接后继集。如果$IMP(i)=\varnothing$,那么称结点i为入口结点;如$IMS(j)=\varnothing$,那么称结点j为出口结点。用path(i,j)表示G中结点i到结点j的一条路径,它本身就是包含结点i和j在内的该条路径上的所有结点集合。在下面的讨论中,结点和任务这两个术语是可互用的。

定义1 如果一个任务已被调度到某一处理器

陈华平 博士讲师,现主要从事并行处理系统和并行程序设计环境的研究。李京 副教授,现主要从事软件构造和面向对象的并行程序设计方法研究。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。

上，那么称该任务是已分配的。如果一个任务 a_i 的直接前趋集 IMP(i)中的所有任务是已分配的，那么称任务 a_i 是就绪的。

2.2 目标机器

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_{ij}],[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,…,P_m}为构成并行结构的一组处理器；(2)[P_{ij}]为 m×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输率，即每单位时间传送的数据量。

2.3 调度

任务图的一个调度其实就是任务图 G 到目标机器的一个映射 f:V→{1,2,…,m}×[0,∞)，f(i)＝(p,t)表示任务 a_i 被调度到编号为 p 的处理器上，起始执行时刻为 t。一般地，我们可用 Gantt 图 GC＝{(P(i),T(i))|i∈V}来直观地表示调度结果，其中函数 P(i)表示分配给任务 a_i 的处理器号，T(i)表示任务 a_i 的起始执行时刻。一般入口结点任务的起始执行时刻以零为计，那么所有任务执行完的那个时刻称为调度长度 SL，它也反映了整个并行程序任务的执行时间，显然 $SL=\underset{1\leqslant i\leqslant n}{MAX}\{T(i)+W_i/S_y\}$，其中 y＝P(i)。

定义2 一个处理器的就绪时刻为该处理器已执行完最近分配给它的一个任务，可以开始执行其它任务的最早时刻，用函数 prdt(j)来表示处理器 P_j 的就绪时刻，用 P_f 表示具有最早就绪时刻的处理器。

3 最优调度算法

这一节我们主要讨论并行分布计算中典型的几种有效最优调度算法。一般说来，这部分算法可分为两类，一类是忽略通讯延迟，而另一类考虑通讯延迟，但必须对通讯模型加以严格的条件限制。

3.1 不考虑通讯延迟

在忽略通讯延迟的情况下，一种最优调度算法是对任务图再加以一些限制条件。如果任务图是树结构的，即任务图中每个任务结点只有一个直接后继，或只有一个直接前趋，并且每个任务的执行时间相同，T. C. Hu[8]给出了线性的最优调度算法，该算法的主要思想是把一个结点到出口结点的路径所经过的边数作为任务优先级，每次首先把当前优先级最高的就绪任务调度分配到 P_f 上。M. Kaufman[9]放宽了上述限制条件，他所给出的最优调度算法允许每个任务的执行时间不同，并且算法执行时间上界为$[1+(m-1)T_{long}]/T_{total}$，其中 T_{long}为最大的任务工作负载，T_{total}为所有任务的工作负载之和。

对任务图的另一种限制是区间有序条件。一般地，可把一个部分有序集(V,<)的每个元素映射到一个实线区间集(R,<)上，设 $l(x)$和 $r(x)$分别表示 V 中元素 x 在 R 中映射区间的左右点位置。

定义3 如果存在一个部分有序集(V,<)到实线区间集(R,<)的映射，并且该映射满足条件：∀ x,y∈V,x<y 当且仅当 $r(x)<l(y)$，那么该部分有序集(V,<)是区间有序的。

上面定义说明了两个相关任务的映射区间不重叠，意味着∀ x,y∈V，要么 IMS(x)⊆IMS(y)，或者 IMS(y)⊆IMS(x)，两者必居其一。区间有序这个特性使得在每个任务执行时间相同的情况下，可应用贪心算法来获得最优调度。C. Papadimitriou[10]给出了该算法，其主要思想是每次选择具有最多直接后继数目的就绪结点，并把它调度到 P_f 上。

还有一类最优调度算法是对目标机器加以限制，最为典型的是在只有两个相同处理单元的目标机器上，如何最优调度忽略通讯延迟并且每个任务执行时间相同的任意任务图。E. G. Coffman[11]给出了一个时间复杂度为 $O(n^2)$的调度算法，其主要思想是通过下面(1)、(2)两个步骤，用一个标记函数 L(*)把集合{1,2,…,n}中的每个元素赋给任务图中的每个任务结点，然后按照标号大小，优先调度分配标号大的任务结点。

(1)一开始把标号1赋给出口结点；

(2)假定标号{1,2,…,j−1}已赋给任务，设 S＝{x|x∈V 且 IMS(x)中结点均已标记}，IMS(x)＝{y_1,y_2,…,y_k}，定义 g(x)为按降序排列的有序集(L(y_1),L(y_2),…,L(y_k))，那么把下一个标号 j 赋给 S 中的一个元素 x′，其中 x′满足条件：∀ x∈S,g(x′)≤g(x)。

3.2 考虑通讯延迟

处理器之间的通讯开销是并行分布计算中不能获得线性加速比的主要因素，一般说来，在考虑通讯

延迟的情况下，调度复杂度与选择的通讯成本模型有很大的关系。首先，对任务图与目标机器施加一些限制条件，主要为任务图中任两个任务间的数据通讯量 D_{ij} 为常数 C_1，每个任务的工作负载为一常数 C_2；目标机器除了全连接假设外，限定相邻处理器之间的传输率为常数 R；并且假定 $C_2/S=C_1/R$，即任务执行时间和通讯延迟相同，设都为单位时间1。在此基础上，常用的通讯成本模型有下列三种[1]：

(1)Model-A：在该成本模型中，表示调度结果的 Gantt 图不反映通讯延迟，但必须保持任务间的优先关系。完成一个并行程序的整个成本 TC＝EC＋CC，其中 EC 为表示执行成本的调度长度，表示通讯成本的 CC 为集合{(u,v)|(u,v)∈E 但 P(u)≠P(v)}的元素个数。

(2)Model-B：该成本模型与 Model-A 类似，只是计算 CC 时更实际些。例如，若 u、v、w∈V 并且(u,v)，(u,w)∈E，P(v)＝P(w)≠P(u)，那么任务 u、v 和 w 间的通讯开销在 Model-A 中要计算两次，而在 Model-B 中只计算一次。所以，表示通讯成本的 CC 为集合{(p,v)|P(v)≠p，但∃ x(x∈IMS(v))，P(x)＝p}的元素数目。

(3)Model-C：该模型假定每个处理单元包含一个 I/O 处理器，因此可同时执行任务和传递消息。对任两个任务结点 i，j∈V，如果 i<j 并且 f(i)＝(k，t)，那么 j 的调度只需满足：若 P(j)＝k，则 T(j)＞t＋1；若 P(j)≠k，则 T(j)≥t＋2。使用该模型可以很容易地把通讯延迟反映到 Gantt 图中，所以可把调度长度作整个成本。

在上述通讯成本计算模型基础上，Afrati 等证明了使用 Model-A，在任意数目处理器的目标机器上最优调度包含考虑通讯的树结构任务图是个 NP 完全问题。M. Prastein[12]证明了使用 Model-B，在只有两个处理器的目标机器上最优调度任意偏序关系的任务图，或在任意数目处理器的目标机器上最优调度树结构任务图均是 NP 完全问题。C. Papadimitriou[10]证明了使用 Model-C，在任意数目处理器的目标机器上最优调度执行和通讯均为单位时间的任务图也是个 NP 完全问题。H. El-Rewini[13]利用增广任务图为中间步骤，基于 Model-C 提出了在只有两个处理器的目标机器上最优调度考虑通讯在内的树结构任务图的有效算法，H. Ali[14]也是基于 Model-C，提出了在任意数目处理器的目标机器上最优调度考虑通讯的区间有序任务图的有效算法。

4 启发式任务调度

既然并行分布计算中一般的任务调度问题均具有 NP 难度，那么如何利用任务图本身所包含的一些启发信息来获得最优调度的近似解，是经常采用的一种技术方法。在共享存储的并行计算模型上，进行启发式任务调度时一般不需要考虑通讯，着重点在于如何获得最大限度的并行性。而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，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[3]。下面我们讨论了常用的几种启发式任务调度方案。

4.1 基于队列结构的启发式任务调度算法 HLS

在并行分布计算中，基于队列结构的启发式任务调度算法[6][15]HLS(Heuristic List Scheduling)是最为常见的一种。它的主要思想是在编译时给任务图中的每个任务按一定规则赋一优先级，并且把当前的所有就绪任务按优先级的大小排列，建立一个就绪任务队列。然后每次从队列中选择优先级最高的任务，按一定规则把该任务调度到合适的处理器(由于通讯开销，该处理器不一定是 P_1)上，并计算出此任务在该处理器上的起始执行时刻，最后获得整个任务图的调度结果。这类启发式算法的思想都很相似，主要区别在于给任务赋优先级的方式有所不同。

定义4 任务图中结点 i 到结点 j 的一条路径的长度 length(i,j)为该路径上包括起始结点 i 和终点结点 j 在内的所有结点的权重之和，即

$$\text{length}(i,j)=\sum_{k\in path(i,j)} W_k。$$

定义5 任务图中一个结点的出口长度为从该结点到出口结点的最大路径长度。用函数 exitlen(i)来表示结点 i 的出口长度。

很明显，一般情况下出口长度大的结点任务应优先执行，因为它很有可能就是影响调度长度 SL 的关键任务。我们把 exitlen(i)作为结点优先级 prio(i)的主要成分，当两个结点的人口长度相同时，可进一步通过它们直接后继结点的数目来决定它们优先级的大小。

定义6 设 prio(i)和 prio(j)分别为两个任务结点 i 和 j 的优先级，如果(1)$exitlen(i)>exitlen(j)$，或者(2)exitlen(i)＝exitlen(j)且|IMS(i)|≥|IMS(j)|，那么 prio(i)＞prio(j)。

如果在共享存储的并行计算模型上执行 HLS

算法，那么反映任务间数据通讯的边权可忽略，所以定义4中每个结点的出口长度主要与路径上的结点权重有关。在分布存储的并行计算模型上，严格地讲，象出口长度这样的启发信息应该包括通讯延迟。但是，任务间的实际通讯延迟不仅是任务图G的函数，也是调度映射f的函数，即把同一结点调度到不同处理器上的通讯开销是不同的。例如，任务a_i与a_j之间的通讯量为$D_{ij}\neq 0$，如果$P(i)=P(j)$，即这两个任务被调度到同一处理器上，那么它们之间的通讯延迟可以忽略，否则，就必须考虑包括竞争开销在内的通讯延迟。为此，可在分布存储的并行模型上近似使用结点的出口长度作为主要启发信息，但在计算任务的起始执行时刻时，必须考虑它与所有前趋任务结点之间的通讯延迟。其实，给一个任务a_i分配处理器时，优先考虑包含IMP(i)结点的处理器，通过把消息传递的源结点和目标结点放在同一处理器上以减少通讯延迟。

定义7 一个任务的消息就绪时刻为该任务已全部接收到其直接前趋所送来的消息的时刻。用函数mrdt(i,j)表示任务a_i被分配到处理器P_j上时的消息就绪时刻。

HLS算法的主要步骤是每次从任务就绪队列中挑选优先级最高的任务为当前任务，然后根据该任务直接前趋的调度情况获得消息就绪时刻，并结合当前各处理器的就绪时刻，以当前任务具有最小起始执行时刻为目标，把该任务调度到适当的处理器上。

4.2 基于任务聚类的启发式任务调度算法TCS

基于任务聚类的启发式任务调度算法TCS (Task Clustering Scheduling)是利用聚类算法为中间步骤，先按一定方法对任务图中的任务进行聚类，以生成一组任务团；然后再把各任务团映射到目标机器上，并对分配到同一处理器上的那些任务进行时序调整，以保持原任务图中各任务之间的先后关系。每个任务团本身由若干个任务组成，同一任务团中的这些任务肯定被分配到同一处理器上。一般可利用聚类任务图中的关键路径、关键任务序列以及原任务图中结点的出口长度等这些启发信息来进行聚类。

聚类任务图CTG=(CV,CE)是在原来任务图G的基础上，把某些任务结点归类合并为一个任务团，并作为结点集$CV=\{C_i|i=1,2,\cdots,k\}$中的一个元素。CV中的每个任务团元素本身是G的一个有向子图，子图中的边保持原来任务图G中任务间的偏序"≺"关系，但是，由于同一任务团中的基本任务被分配到同一处理器上，所以同一任务团内原基本任务结点间的边权忽略为零。由于我们主要关心的是一个任务团由哪些基本任务组成，而任务间的先后关系是始终保持与原任务图一致的，所以就简单地用$C_i=\{a_{i1},a_{i2},\cdots,a_{in}\}$来代表第i个任务团，用$a_j\in C_i$表示任务$a_j$属于第i个任务团。很明显，任务图G本身也是一个特殊的聚类任务图CTG_0，CTG_0的每个结点为$\{a_i\}(1\leqslant i\leqslant n)$，即每一基本任务为一个任务团，$CTG_0$的边集CE为原任务图G的边集E。

我们的目标就是要找一个最优聚类，使得按它所得到的聚类任务图进行调度时可获得最小的调度长度。但遗憾的是，求任意任务图的最优聚类问题本是一个NP问题。为此，经常使用的是一种非回溯的启发式算法，它的主要思想是从初始聚类任务图CTG_0开始，根据当前聚类任务图一些启发信息，按照一定的目标函数执行一系列的聚类求精步操作，直到达到一定目标为止。非回溯思想主要体现在一旦在某一聚类求精步中两个任务被合并到同一任务团中，那么在以后的操作中这两个任务不再分开。

在每一聚类求精步中所使用的主要操作为"归零操作"，它的主要思想是按照一定规则选择CE的子集E_r，把这些边相连的两个任务团分别合并成为一个任务团。由于合并后边两端的基本任务属于同一任务团了，所以它们之间的边权改变为零。根据以上讨论可知，任务聚类关键在于选择E_r，我们就简单地用ref_r来表示每一步的求精操作，它主要是把通讯矩阵中的某些元素修改为零，所以它也体现了通讯矩阵的变化。

定义8 一个任务聚类问题可表示为聚类任务图的状态转换序列$(CTG_0,ref_1,CTG_1,ref_2,CTG_2,\cdots,ref_s,CTG_s)$，其中$ref_r(1\leqslant r\leqslant s)$为第r步聚类求精操作，它的主要动作就是对$E_r$中的边进行"归零操作"，即$D[ij]_{r-1}\xrightarrow{ref_r}[Dij]_r$。

定义9 一个聚类任务图的关键路径CP为该聚类任务图中具有最大路径长度的那条路径。关键任务序列CTS或者是聚类任务图的关键路径上的任务组成的序列，或者是其执行长度为调度长度的一个任务序列。

S. Kim和J. Browne[7]提出了以带权路径长度为目标的聚类算法，它的主要思想是以$F(CTG_r)=\omega_1*\sum INS(a_i)+(1-\omega_1)(\omega_2*\sum D_{ij}+(1-\omega_2)*$

$\sum D_i^r$)为目标函数，其中的$\sum D_i^r=\{weight(a_i,a_j)\mid a_i\in CP_r$ 并且 $a_i\notin CP_r$，或者 $a_i\in CP_r$ 并且 $a_i\notin CP_r\}$，它反映了由 CP_r 上任务所组成的任务团与其它任务团之间的通讯开销。ω_1和 ω_2是介于0与1之间的调整因子，一般说来，如果目标机器中每个处理单元的执行速度较快，而通讯传输较慢，那么数据通讯对并行执行时间的影响较大一些，这时应把 ω_1调小一些，反之，则应调大一些。对于 ω_2，如果在求路径 CP_r 时不仅考虑 CP_r 上的通讯开销，还要多考虑一些新生成的任务团与其它任务团之间的通讯开销，那么可选择小一些的 ω_2。当 $\omega_1=1/2,\omega_2=1$时，目标函数 $F(CTG_r)$就主要体现关键路径长度这个参数。

V. Sarkar[16]提出了按边权非增序进行归零操作的聚类算法，它的主要思想是优先考虑那些通讯量较大的边，因为在通讯延迟较大的分布存储并行计算环境下，任务之间的数据通讯非常有可能成为影响整个并行程序执行时间的瓶颈，所以该算法把通讯量较大的一组任务结合到同一任务团中。

A. Gerasoulis[17]提出了以关键任务序列为目标的聚类算法。它的主要思想是在每一聚类求精步中识别出当前聚类任务图 CTG_{r-1}的关键任务序列，然后按一定方法选择该任务序列上的某条未标记边 e_r，使得对该边进行归零操作后，所新产生的聚类任务图的调度长度 SL_r 小于或等于原来聚类任务图的调度长度 SL_{r-1}。选择 CTS_r 中归零边 e_r 的方法很多，如可从上到下选择 CTS_r 中第一条未标记的边，或者选择 CTS_r 中具有最大边权的未标记边。

M. Y. Wu 和 D. Gajski[18]提出了以任务具有最早起始执行时刻为目标的聚类算法，该算法并不是一开始就把原始任务图 G 中的每个基本任务作为一个任务团，然后通过反复的归零合并操作把某些任务图合二为一。这个算法的主要思想是利用每个结点的出口长度作为任务优先级的主要部分，按优先级的非增次序逐个把未标记的就绪任务结点合并到具有最早起始执行时刻的任务团中[6]。

按照上面算法求得任务团集 CV 后，还必须把 CV 映射到具体的目标机器上。如果任务团个数大于处理器数 m，那么必须合并某些任务团，使得整个任务团个数等于 m。另外，如果目标机器不是全连接的，那么应把任务团之间通讯量较大的那些任务团尽量映射到相邻的处理器上，这些操作均是在编译时静态完成的。

4.3 任意目标机器上的启发式任务调度算法 HTS

上面两类启发式任务调度算法均对目标机器进行了全连接和每个处理单元执行速度相同的假设，它们只考虑了数据传输所引起的通讯延迟，而忽略了非全连接拓扑结构中由于通讯中继和通讯线路竞争引起的通讯开销。但是，随着当今 MPP 系统的发展，特别是 PVM 与 NOW 系统的广泛应用，分布存储环境下的各种通讯开销已成为影响并行程序执行性能的一个主要因素。这一节我们主要讨论任意目标机器上的启发式任务调度算法 HTS(Heuristic Task Scheduling)。

设 T_{ij}表示任务 a_i 在处理器 P_j 上的执行时间，那么 $T_{ij}=(W_i/S_j)+B_j$。设每个处理器的消息传递启动时间为 I，每两个相邻处理器之间的消息传输率为 R，用 $C(i_1,i_2,j_1,j_2)$表示处理器 P_{j1} 上的任务 a_{i1}与处理器 P_{j2}上的任务 a_{i2}之间的通讯开销，用 H_{j1j2}表示处理器 P_{j1}到处理器 P_{j2}的线路条数(也称之为中继段数)，用 CD_{j1j2}表示 P_{j1}与 P_{j2}之间通讯线路的竞争开销，那么 $C(i_1,i_2,j_1,j_2)=((D_{i1i2}/R)+I)*H_{j1j2}+CD_{j1j2}$，其中 H_{j1j2}和 CD_{j1j2}与当前的调度通讯情况有关[19][1]。

设 fint(x,y)表示任务 a_x 被分配到处理器 P_y 上时的完成时刻，即 $fint(x,y)=T(x)+T_{xy}$，消息就绪时刻 mrdt(i,j)主要由任务 a_i 直接前趋的 fint 函数值及其通讯开销所决定。

$$mrdt(i,j)=\underset{x\in IMP(i)}{MAX}(fint(x,P(x))+C(x,i,P(x),j)) \quad (I)$$

只有到达 mrdt(i,j)时刻，任务 a_i 才能在 P_j 上接收到全部所需消息，至于到该时刻能否开始执行，还要看 prdt(j)的大小。求得 mrdt(i,j)后，可进一步确定 fint(i,j)的大小，其计算公式为：

(1)如果 $IMP(i)=\varnothing$，那么 $fint(i,j)=prdt(j)+T_{ij}$

(2)如果 $IMP(i)\neq\varnothing$，那么 $fint(i,j)=\max(mrdt(i,j),prdt(j)+T_{ij})$ (II)

由于每个处理器的执行速度不尽相同，所以每个任务在不同处理器上的执行时间不尽相同，因而具有最小完成时刻的执行处理器并不一定是具有最早执行时刻的处理器。HTS 算法[19]就是每次从 RTQ 中取出队首任务作为当前分配任务 a_c，并以获得最小的完成时刻为目标，把使 $fint(c,*)$取得最小值的处理器号作为该任务的执行处理器。然后交替使用递推公式(I)(II)，以求得 G 中每个任务的执行处理器号和相应的起始执行时刻。

为了比较精确地估计由竞争引起的开销，HTS算法为每个处理器 $P_i(1\leqslant i\leqslant m)$ 建立一个选路表 RTB_i，该选路表由 $m-1$ 条记录 $R_j(1\leqslant j\leqslant m$ 且 $j\neq i$，下同）组成，每条记录记载了 P_i 对其它处理器的当前通讯状况。每个记录由 H、L 和 C 三个域组成。为了简单起见，我们用 H_{ij} 表示处理器 P_i 到处理器 P_j 的中继段数；$L_{ij}(1\leqslant L_{ij}\leqslant m)$ 是个处理器号，表示处理器 P_i 给处理器 P_j 传递消息时，P_i 准备选择使用的每一个中继处理器号，也即表示了 P_i 准备使用的通讯线路；C_{ij} 表示当前 P_i 给 P_j 传递消息时，由于竞争而引起的通讯开销。初始时，HTS 算法把 CD_{ij} 均置为零，并根据拓扑邻接矩阵 $[P_{ij}]$，利用处理器两两之间的最短路径求得 H_{ij} 和 L_{ij}。如果两点间多于一条最短路径，那么任选一条作为 H_{ij} 和 L_{ij} 的初值计算依据。

选路表主要用于获得传递一个消息的最佳路径，以及通过竞争延迟估计，来更精确地选择执行某一任务的最佳处理器。在调度过程中需要不断地修改这些选路表，这样就能根据当前的通讯交通情况来选择适当的传送线路和执行处理器。一般说来，对这些选路表的修改操作频繁些，那么对整体通讯交通情况就更了解得清楚全面些，但这些表操作的时间复杂度也就要高一些，这两者之间本身有一个折衷[19]。（未完待续）

参考文献

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

[2] J. Ullman, Np-complete scheduling problems, J. Computer and System Sciences,10',1975

[3] T. Lewis et al., Introduction to parallel computing, Prentice Hall,1992

[4] T. Casavant et al., A taxonomy of scheduling in general purpose distributed computing systems, IEEE Trans. Soft. Eng. ,14(2)1988

[5] V. Saletore, A distributed and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks, Proc. DMCC-5, Portland, Oregon,1990

[6] A. F. Bashir et al., A statical study of a task scheduling algorithm, IEEE trans. Comput. ,C-32 (8)1983

[7] S. Kim and J. Browne, A general approach to mapping of parallel computation open multiprocessor architectures, Proc. Intl. Conf. on Parallel Processing,1988

[8] T. C. Hu, Parallel sequencing and assembly line problems, Operations Research,9(6)1961

[9] M. Kaufman, An almost-optimal algorithm for the assembly line scheduling problem, IEEE Trans. Comput. ,c-23(11)1974

[10] C. Papadimitriou et al., Scheduling interval-ordered tasks, SIAM J. Computing,8,1979

[11] E. G. Coffman, Computer and job-shop scheduling theory, John Wiley & Sons,1976

[12] M. Prastein, Precedence-constrained scheduling with minimum time and communications. M. S. Thesis, University of Illinois at Urbana-Champagin,1987

[13] H. El-Rewini and H. Ali, On the scheduling problem with communication, Technical Report, University of Nebraska at Omaha,1993

[14] H. Ali and H. El-Rewini, The time complexity of scheduling interval orders with communication is ploynomial. Parallel Processing Letters,3(1)1993

[15] T. L. Adam et al., A comparison of list schedules for parallel processing systems, C ACM,17,1974

[16] V. Sarkar, Partitioning and scheduling parallel programs for execution on multiprocessors, MIT Press,1989

[17] A. Gerasoulis and T. Yang, A comparison of clustering heuristics for scheduling DAGs on multiprocessors, J. Parallel and Distributed Computing, 1992

[18] M. Y. Wu and D. Gajski, A programming aid for hybercube architecture,J. Supercomputing,2 1988

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