国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第16卷第1期 1994年3月 Vol. 16 No. 1

任务划分与处理机调度的递归－分治方法*

曹介南 杜铁塔

（国防科技大学电子计算机系 长沙 410073）

摘 要 本文介绍并实现了一种如何把一个顺序执行的任务集，根据其子任务之间潜在的并行性，划分成若干个可并发执行的任务子集，并把每个子集分配给一个处理机，使各处理机之间的数据通信量尽可能地少，同时兼顾各处理机之间负载平衡的算法。最后给出了几个典型例题的试算结果，为了满足用户的不同要求，文章还提出了几点改进方法。

关键词 多任务划分，处理机调度，机间通信，负载平衡，递归－分治，受益值

分类号 TP311

多任务划分与处理机调度是指把一个计算问题（任务集）分解为多个可并发执行的任务子集，并把每个子集映射到某个处理机上并行执行，以加快问题的求解速度。除非任务之间具有天然的类聚特征（如例2中所示的那样），否则任务划分算法都难保证各处理机之间没有通信，而在多机系统中，机间通信与同步往往是提高问题求解速度的瓶颈问题，因此，在任务分划过程中，如何把相互之间的通信量多的子任务尽可能地安排到同一处理机上，以减少子任务之间的通信量成为关键性的问题。同时，还必须考虑各处理机之间负载平衡问题，而二者又是相互矛盾的。不过，对于不同的多机系统，所追求的目标是不同的。例如，对于具有共享存贮器的多处理机系统，由于实现了紧耦合互连，机间通信时延较小，调度算法应主要考虑系统的吞吐率及负载平衡问题；而对用多台计算机构成的多机系统，计算机之间通过链路进行通信（即松耦合系统），通信时延较大（包括传送、路由选择、通信的同步等开销），而处理机速度较快，通信成为主要开销，任务划分时减少机间通信量成为首要的问题。下面将要介绍的递归－分治算法正是为了解决这类问题而设计的。所谓递归－分治，就是分而治之，即把一个难解的大问题分成多个体积接近的子问题，如果子问题能容易地求解，则个个击破之，否则依次对每个子问题递归用分治法求解之。

在设计多任务划分算法时，还有许多其它因素值得考虑，例如任务的粒度即每个任务的计算量或权，它取决于计算问题本身，任务粒度不能太大，象单机上一样整个问题就是一个任务，缺乏并行性；也不能太小，否则任务之间的通信就会增多反而影响效率。通常可以让几个计算量小的语句合并成一个任务或者把一个计算量大的语句分成若干步

* 1992年7月8日收稿

来执行，每步定义为一个任务，原则是让各任务的计算量大致相等。另一个因素是划分粒度，它定义为组成系统的计算机或处理机台数。有一个现象值得注意的是，对松耦合系统，有些任务并不是处理机台数越多，它完成的速度越快，因为如把任务均匀地分到所有处理机上，势必导致各处理机间通信频繁，通信时间增多，反而使任务完成时间推迟，还不如把任务按通信量尽可能小为原则分配到几个而不是全部处理机上。当然这里有个度量问题，即当通信量（次数）N_{com}大到多少时就不再分割下去了。这主要取决于处理机的运算速率V_{run}和机间通信速率V_{com}及任务本身的计算量W_t，
一般应满足：

$$N_{com} \leqslant 0.5 * \frac{W_t}{V_{run}} * V_{com}$$

例如：计算量$W_t=40$，$V_{run}=10$，$V_{com}=1$，则当把该任务集划分成两块时通信次数应满足$N_{com} \leqslant 2$次，否则划分就不合算。

当任务分割成若干个子集后，剩下的问题就是把各子集映射到处理机上去，映射的原则是子集之间通信量最大的一对应该安排到具有最短路径的两个处理机上，以进一步减少因转发而带来的通信开销。

当然，还可以采取一些其它技术来减少通信与同步开销，例如，引进冗余计算以减少通信次数，即如果某变量的求值时间小于把它传送到另一个处理机的通信时间，且该变量被分在其它处理机上的任务多次引用，则可考虑把它分布在各个处理机上作冗余计算，以避免机间通信。又如采取“选择最佳通信时机”（见参考文献[2]）可以有效地减少同步开销等等。

1 算法描述

递归一分治算法的实质是把任一矩阵通过行列初等变换，转换成块对角矩阵，使块与块之间的联系最少，即块对角以外的非零元素呈稀疏状，同时保证块的大小大致相同。算法的基本思想是，把一个任务集合GT划分成二个相对独立的子集GL和GR，使GL和GR中元素之间的相关性最小，并尽可能使GL和GR中任务的计算量相同。再对GL和GR递归地作上述划分，直到递归深度等于最大递归深度$\log_2 Pn$(Pn为处理机台数)为止，最后将得到Pn个子集，然后把每一子集分配给某一个处理机，下面为算法的形式描述：

Algorithm Sheduling (GT, GP, Map)；{主程序}
{GT：输入的任务相关矩阵，具有tn * tn个元素，对角线元素表示各任务的计算量wt，非对角线元素GT[i][j](0≤i, j<tn, j≠i)表示任务i与任务j之间的通信量。
GP：是输入的处理机邻接矩阵，GP[i][j]=1(0≤i, j<Pn, i≠j)表示处理机i与j之间有物理链路；
Map：输出，任务集T到处理机集P的一个映射（不一定为一对一映射），Map[i]=j表示第i任务被分配到处理机j上计算，(0≤i<tn, 0≤j<Pn)，tn为任务集T中任务的个数；
Pn为处理机集P中处理机台数

```
}
BEGIN
1) maxdepth=log2Pn;                    {最大递归深度}
   depth=0;                            {当前递归深度}
   cluster[i]=0; 1≤i≤tn                {初值为0的数组；当从子程序DIVDE_RECURSIVE返回后，cluseter的元素都被分成Pn组}
2) DIVDE_RECURSIVE (GT, tn, depth, cluster); {调用分治子程序}
3) ALLOCATE_PROCESSORS (cluster, GP, Map);
   {cluster中每一组映射到GP中的一个处理机上，考虑到物理链路的情况下，使机间的通信代价最小}
END
Algorithm DIVDE_RECURSIVE (Gn, n, depth, cluster)
{递归一分治子程序}
BEGIN
  if (depth≠Maxdepth) {如果达到最大递归深度则返回，否则做下面的工作}
    BEGIN
    MINCUT (Gn, n. Lcount, GL, Rcount, GR);
      {把Gn中的n个任务分成左、右两个子集GL和GR，大小为Locunt和Rcount，使它们之间的通信次数尽可能地少，且计算量大致相等}
    DIVDE_RECURSIVE (Lcount, GL, depth+1, cluster);
                {对左子集再递归一分治}
    DIVDE_RECURSIVE (Rcount, GR, depth+1, cluster);
                {对右子集再递归一分治}
    END
END
Algorithm MINCUT (Gn, n, Lcount, GL, Rcount, GR);
  {划分子程序}
  BEGIN
  1. 把Gn中的节点随机分成计算量大致相等的两部分GL和GR
  2. 标示所有顶点为“未处理”，lock[i]=0; 0≤i≤n
     为每个顶点设立一个“受益值”gain[i]=0; 0≤i≤n
  3. REPEAT
       BEGIN
       3.1 CALCULATE_GAIN_WT ();
        {为每个顶点V计算受益值gain[v];
         gain[v]=当顶点v从当前子集中移到另一个子集中时两个子集之间通信次数的减少量（可能为负数）;
         为GL, GR计算权W1, W2，并保存其差之绝对值于平衡数组中：
```

balance[0]=abs(W1−W2])}

3.2 step=0;

3.3 REPEAT

BEGIN

3.3.1 用 f（f=L 或 R）标示具有较大权之子集；

3.3.2 设 V 为 Gf 中具有最大受益值 gain（为正）的未处理过的顶点；
如 Gf 中无这样 V（即所有顶点均处理过了）则退出 3.3 循环；

3.3.3 假设 V 被移到另一个子集中，为 GL，GR 中所有未处理的顶点重新计算 gain，weight，balance[step]=W1−W2；CALCULATE_GAIN_GT ()；

3.3.4 置 V 为已处理标志：lock[V]=1；
记录 V 移动状态：movnode[step]=V；
累计 gain：Sgain[step]=Sgain[step−1]+gain[V]；

END

UNTIL(step>n)

3.4 选择最大的 K，1≤K≤step 使：
G=Sgain[k]=MAX(Sgain[1],Sgain[2],…,Sgain[step])；
在选 K 时，如果 Sgain[i]=Sgain[j],j>i 且 balance[i]>balance[j]，则应选取 K=j，即 j 时刻的负载平衡性好于 i 时刻。

3.5 if ((G>0) OR ((G=0) AND (balance[k]<balance[0])) then
根据 movnode 从第一步到第 K 步真正执行所有的移动；

END

UNTIL ((G<0) OR ((G=0) AND (balance[k]>balance[0])))

4. RETURN；

END

2 处理机调度算法

处理机调度算法类似于 MINCUT，其形式描述如下：

Algorithm ALLOCATE_PROCESSORS (cluster，GP，Map)；

BEGIN

1. 生成 Map 的初始映象：把 cluter 每一组任意分配给一处理机；
2. 置所有处理机节点为未处理：lock[i]=0；0≤i≤Pn
3. REPEAT

BEGIN

3.1 对所有处理机对(Pi，Pj)，假设 Ci，Cj 两个任务子集分别被分在 Pi，Pj 处理机上，交换一下 Ci，Cj 的分配关系为 (Pi，Pj) 计算交换后带来的通信方面受益值 gain.

3.2 step=0；

3.3 REPEAT

BEGIN

3.3.1 step=step+1;

3.3.2 选出未处理的具有最大 gain 值的处理机对(Pk，Pe)；如果无这样的对存在（即所有处理机均处理过了），则退出 3.3 循环；

3.3.3 假设 pk，pe 上的子集被交换，为所有未处理的处理机修改 gain 值。

3.3.4 标示 Pk 和 Pe 已处理：lock[Pk]=lock[Pe]=1；

END

UNTIL（不存在未处理的处理机）

3.4 选择最大的 K，1≤K≤step 使：

$$G = \sum_{i=1}^{K} gain[i] \text{ 最大；}$$

3.5 IF（G>0）then 从 step 1 到 step k 对 MAP 执行所有的交换；

END

UNTIL（G≤0）

4. 返回 MAP 的值。

END

3 算法的几点改进

由于用户的要求不同，有的追求计算时间最短，有的追求解最优等等，为此对本算法的核心 MINCUT 子程序（它基于 Kernighan-Lin 的最小割集算法）稍加改变即可满足这些要求。

（1）在 MINCUT 的 3.3.2 处，如果 V 不是从权最大的子集中选取，而是从整个集合中选取，设置一个平衡门拦 gate，假设当 V 移到另外一个子集中去后负载平衡度不低于该门栏值的话才执行 V 的移动，否则本次不移动。这样可以进一步降低通信次数，更接近问题的最优解。

（2）在 MINCUT 的 3.3.4 处，如果不立即标示 V 为已处理，而是设立一个移动计数器，当 V 被移动 m 次后才设定它为已处理（一般取 m=3 较合理），这样也更接近最优解。当然算法的执行时间也随之增加（下面的例题就是用这种方法求得的）。

（3）在 MINCUT 的 3.3.2 处，如果每次从 GL 和 GR 中各选一个 V1 和 V2，交换一下它们的位置，实验证明这样通信次数不增加，而算法的执行时间可减少近一半，且负载平衡性更好。

（4）在 MINCUT 的 3.3.2 处，在选 V 的同时，选一个 NV=受益值为负且最小的顶点，标示它别移动，这样也可改善算法的执行时间。

（5）把以上几点组合起来可有效地提高算法和效率和求解精度。

（6）本算法是以邻接矩阵为输入的，如子任务的个数较多，应以邻接边为输入，读者不难改进本算法。改进后的算法可作为任何多机系统中任务划分与处理机调度的算法。

4 性能评价

我们在 Tuboc C 环境下实现了该算法，并试算了许多典型例题，结果说明与目前公认为很有效的关键路径 tcp 法（另一种启发式算法）类似，它们都接近处理机调度问题的最优解。结果见打印清单。

算法的执行时间 T(n)随递归的深入呈指数下降，因此算法主要时间耗费在递归进入 MINCUT 子程序的 calculate-gain_gt 中，语句 3.1 只执行常数次（1～2 次），它是$C2*n^2$的，语句 3.3 执行 n 次，它是$C1*n^3$的，故有：

$$T(n)=\begin{cases} C0 & n=1 \\ 2T(n/2)+C1*n^3+C2*n^2 & n>1,\ Pn>1 \end{cases}$$

其中 C0，C1，C2 为常数，解此递归方程（设$n=2^k$）；

$$\begin{aligned} T(2^k)&=2T(2^{k-1})+C1*n^3+C2*n^2 \\ &=\cdots\cdots \\ &=2^kT(1)+C1(2^{3k}+2^{3(k-1)}+\cdots+2^{3(k-(k-1))}) \\ &\quad\ C2(2^{2k}+2^{2(k-1)}+\cdots+2^{2(k-(k-1))}) \\ &=n*C0+C1*n^3+C2*n^2+C \qquad (C\text{ 为常数}) \\ &=O(n^3) \end{aligned}$$

所以算法的执行时间$T(n)=O(n^3)$级的。

附 1 典型例题试算结果（矩阵中‘·’表示 0，‘*’表示 1）：

例题 1 输入矩阵（其任务相关图如右）看似无规律，但经划分后，块之间仅有四次通信。

The input graph is:

```
* · · · · · * · · · * · · · ·
· * · · · * · · · * · · · * · ·
· · * · · · · · · · * * * * *
· · · * · * · · · * · · · * · ·
· * · · · * · · · * · · · * · ·
· · · * · * · · · * · · · * · ·
* · · · · · * · * · · · * · · ·
· * · · · · · * · · · · · * · ·
· · · · · · * * · · * · * · · ·
· · · * · * · · · * · · · * · ·
· · * · · · · · · · * * * · · ·
* · · · · · * · * · · * · · · ·
· * * · · · · · · · * · * · · *
· · · * · · · · · * · · · * · ·
· * · · · * · · · · · · · * · ·
· · * · · · · · · · * · * · · *
```

This is output

```
* * * * · · · · · · · · · · · ·
* * * * · · · · · · · · · · · ·
· * * * * · * · · · · · · · · *
* * * * · · · · · · · · · · · ·
· · · · * * * * · · · · · · · ·
· · · · * * * * · · · · · · · ·
· · · · * · * · · · · · · · · ·
· · · · * * * * · · · · · · · ·
· · · · · · · * * * * · · · * ·
· · · · · · · · * * · · · · · ·
· · · · * · · · * * * * · · · ·
· · · · · · · · * * * * · · · ·
· · · · · · · · · · · · * * * *
· · · · · · · · · · · · * * * *
· · · · · · · · · · · · * * * *
· · · · · · · · · · · · · * · *
```

The number of processor = 4

processor 0= 0 6 8 11

Processor 1= 1 4 7 14

rpocessor 2= 2 10 12 15

processor 3= 3 5 9 13

（负载绝对平衡）

the ncluster grapth is:

11 1 0 1

0 11 0 0

0 1 11 1

0 0 0 11

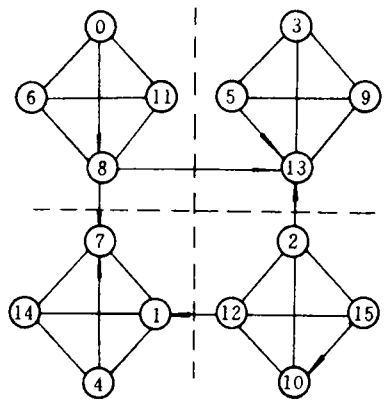


图 1 任务相关图（无箭头边为双向边）

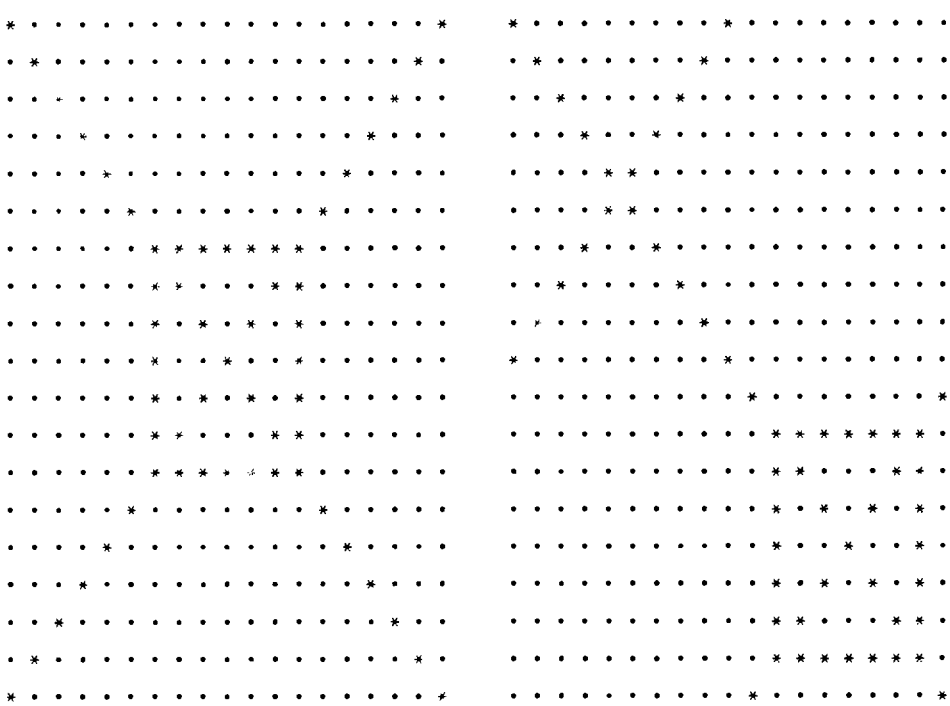
the number of comm. in processor=44

the number of comm. between processor= 4

例题 2 输入矩阵已是对角矩阵，经划分后，还是对角矩阵，不改变原有性质，算法无付作用（图略）。

例题 3 输入矩阵为双对角矩阵，经划分后，可以看出结果是最佳的。

The input graph is: this is output:



The number of processor= 2

processor 0= 0 1 2 3 4 14 15 16 17 18

processor 1= 5 6 7 8 9 10 11 12 13 （负载基本平衡）

the cluster grapth is:

10 0
0 28
the number of comm. in processor=38
the number of comm. between processor= 0

参 考 文 献

1 P sadayappan, Fikret Ercal. Cluster-Partitioning Approaches to Mapping Parallel Programs onto A Hypercube

2 杜铁塔. 同构型并行仿真机系统结构及其软件支撑环境的研究与实现：[学位论文]. 长沙：国防科技大学

3 曹介南. 连续系统仿真语言在同构型多机系统上的研究与实现：[学位论文]. 长沙：国防科技大学

The Method of Recursive Divide for Task Partition and Processor Scheduling

Cao Jienan Du Tieta

(Department of Computer Science, NUDT, Changsha, 410073)

Abstract

In this papel, we present an efficient method for task partition and processor scheduling By the method we can divide a set of sequential tasks into many subsets, which can be performed parallel in its processor. Both the communication between the processors approaching to the low critical of the problem, and the balance of the load in processor are good.

Key words task partition, processor scheduling, communication between the processors, balance of load, recursive_divide, gain