基于多处理机系统的最短路径并行算法的高效实现①

Realization of the Shortest Path Parallel Algorithm Based on Multi-Processor System

唐俊奇　(湄洲湾职业技术学院 福建 莆田 351254)

摘　要： 用图论的方法对最短路径问题进行数学描述，分析了单源最短路径 Moore 和 Dijkstra 两种算法对多处理机系统在图的搜索阶段的并行性差异；提出了 Moore 算法在多处理机系统中高效并行实现的两种切实可行的方案。

关键词： 图论的方法 最短路径问题 多处理机系统 算法

1　引言

最短路径问题在现实中得到极其广泛的应用，如：汽车电子导航系统以及各种应急系统等，它们都要求以最快的速度计算出到出事地点的最佳路线，在行车过程中还需要实时计算出车辆前方的行驶路线；我们可以把它们概括为一个数学模型：求最短路径问题的模型；模型是对现实世界的高度概括. 模型自身的概括性不可避免地在模型和现实之间产生了一定距离，缩短这一距离的方法是科技进步过程中长期而不懈的努力。当今虽然计算机 CPU 的主频随着时间的推移也在不断提升，计算机的系统性能也在不断改善，但是单处理机的性能对许多复杂的最短路径问题的求解已不能满足现实的需求。因此，多处理机系统可以克服单处理机在处理速度方面的不足。在多处理机系统中如何提高的求解最短路径问题计算速度是摆在我们面前的一大课题。

2　最短路径问题数学描述[1]

2.1 带权图

用图论的方法把最短路径问题进行数学描述。在用图论研究实际问题时，除了将实际问题转化成为无向(或有向)图外，有时还将图的边(或顶点)附加上一个实数，如果一个图的所有边都附加上一个实数，这个图就成了带权图了。

定义：设无向图 G＝〈V，E〉，对于 G 中的任意一条边(V_i，V_j)，都存在实数 $W_{i,j}$ 与之对应，称 $W_{i,j}$ 是边(V_i，V_j)的权，G 连同边上的权称为带权图，有时记作 G〈V，E，W〉，其中 W 是边集 E 到实数集的函数。

类似地定义有向图边上的权及有向带权图。

这里所说的权是指与边有关的数量指标，根据实际问题的需要，可以赋予它不同的含义，例如表示距离，时间，费用等。

将上述问题化为图论问题就是：在有向带权图 D＝〈V，E，W〉或无向带权图 G＝〈V，E，W〉中，给定始点 V1 和终点 VP，求从始点 V1 到终点 VP 的一条路径，使路径的权最小，这就是最短路径问题。

1.2 带权图的建立

我们用“爬山的最好路线”作为来做实验：

给定一组互连结点，结点间的链路用 “权值”标记，求出从一指定结点到另一指定结点的路径，要求该路径的累计权值最小。

互连结点可用图来表示。按照图的术语，结点称为顶点，链路称为边。如果边隐含着方向(即边只能沿一个方向经过)则该图为有向图。要求解的问题是搜索图中最短路径之一。

图 1 所示，对应的有向图如图 2，其中权值表示通过两相邻营地间路径所花费的代价。注意，本文中的图是有向图，权值与沿一特定方向通过路径有关。理论上，我们可以两个方向作出所有营地间的路径，即全连通图，不过由于每个方向的权值会不一样它仍

① 收稿时间:2009-02-04

是一个有向图。一个方向的代价会与相反方向的代价不同(下山而不是上山！)在有些问题中，两个方向的权值是一样的比如，寻求开车的最短路径时，两个方向的距离是一样的，权值相等，因此可用一个无向图表示。

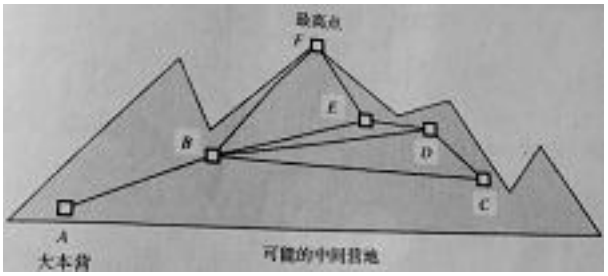


图 1　爬山示意图

我们首先要建立一种在程序中表示图的方法。从顺序编程中我们已知道，在程序中图可以有两种基本方法表示：

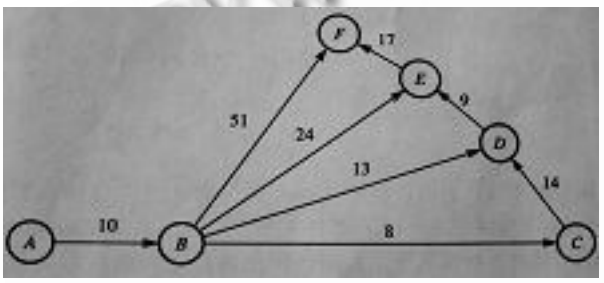


图 2　爬山有向图

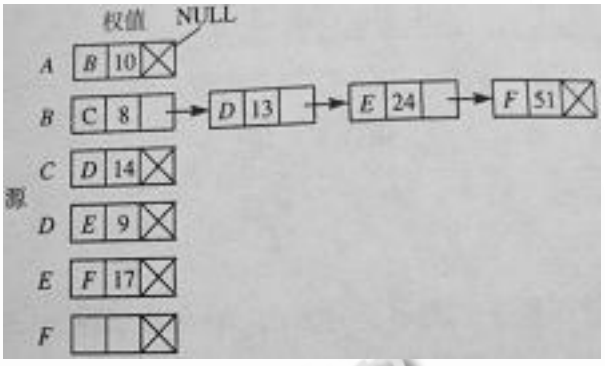
1) 邻接矩阵：一个二维数组 a[i][j]存放与顶点 i 和 j 之间的边(如存在)有关的数值。

2) 邻接列表：每一个顶点，有一个通过边和相应权值直接连接到该顶点的顶点列表。

对于爬山问题，我们可以用如图 3 所示的邻接矩阵和邻接列表两种方法表示。邻接列表是用链表实现的。邻接列表中边的顺序是任意的，选定某一方法应依赖于图的特征和程序的结构。对于顺序程序来说，通常邻接矩阵用于稠密图，即图中每个顶点都有很多边，而邻接列表主要用稀疏图，因为图中每个顶点的边很少。两者的差别在于对空间(存储)需求的不同，邻接矩阵的空间需求的不同，邻接矩阵的空间需求为 $O(n^2)$，邻接表的空间需求为 $O(n\nu)$，其中每个顶点有 ν 条边，而总共有 n 个顶点。一般每个顶点的的 ν 是不同的。因此，邻接列表空间需求的上界为 $O(n\nu_{max})$。访问邻接列表比访问邻接矩阵慢，因为需要顺序遍历邻接列表，这可能要 ν 步。

	A	B	C	D	E	F
A	∞	10	∞	∞	∞	∞
B	∞	∞	8	13	24	51
C	∞	∞	∞	14	∞	∞
D	∞	∞	∞	∞	9	∞
E	∞	∞	∞	∞	∞	17
F	∞	∞	∞	∞	∞	∞

a)　邻接矩阵



b)　邻接列表

图 3　图的表示

除了空间和时间特征外，对于并程序需要考虑任务的划分及对信息访问的影响。下面我们假定采用邻接矩阵表示(尽管该图是稀疏图)。

2　在多处理机系统中进行图搜索的算法选择[2]

在图 1 中，我们可以看出：因为达最高点只有几种方式，因此搜索也相当简单；但在更复杂的问题中，如汽车电子导航系统的搜索就不这么容易管理，要在多处理机系统中对图进行高效的搜索就必须使用合适的算法。现有两个著名的单源最短路径算法，它们都是用求出一从个源顶点到一个目的顶点的最小权值累计来确认到达最高点的方式，这两种方法分别是：

Moore 的单源最短路径算法(Moore，1957)

Dijkstra 的单源最短路径算法(Dijkstra ,1959)

虽然这两种算法在许多方面是相类似的。但是它在多处理机系统中进行图搜索时，这两种算法由于并行性的不同将会产生很大的差异。下面我们来讨论哪种算法更适合在多处理系统中进行图搜索。

2.1 Moore 算法[3]

由源顶点开始，在考虑顶点 i 时，其基本算法为：找出经过顶 i 到顶点 j 的距离，并与当前点到顶点 j 的最小距离比较，如经过顶点 i 的距离更短，则改变最小距离；用数学记号表示即是，如果 d_i 是从源顶点到顶点 i 的当前最小距离，$w_{i,j}$ 是从顶点 i 到顶点 j 边的权值，则有：

$$d_j = \min(d_j,\ d_i + w_{i,j})$$

图 4 对该算法做了说明。从(Bertsekas and Tsitsiklis，1989)中可以看出，通过简单地重复使用上述公式就可以解决问题(一个迭代解法)。

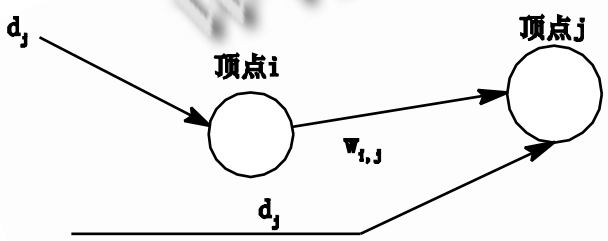


图 4　Moore 最短路径算法

该公式用有向搜索实现。我们需构造一个先进先出的顶点队列，它用来存放要检查的顶点列表，只有在顶点队列中顶点才会被考察，开始时只有源顶点在队列中；另外还需要一个结构以存放从源顶点到其他每个顶点的当前最小距离。假定有 n 个顶点，且顶点 0 是源顶点，从源顶点到顶点 i 的当前最小距离存放在数组 dist[i](1≤i<n)中。开始时国为并不知道这些距离，故数组元素的初值为无穷大。假定 w[i][j]存放从顶点 i 到顶点 j 的边的权值（无边则为无穷大），代码可为如下形式：

```
newdist_j = dist[j] + w[i][j];
if (newdist_j< dist[j])   dist[j] = newdist_j ;
```

当找到到顶点 j 的较短距离后，把顶点 j 加入到队列(如还不在队列)这会使顶点 j 被再一次检查，这是 Moore 算法的一个重要特征，在 Dijkstra 算法中没有这一特征。

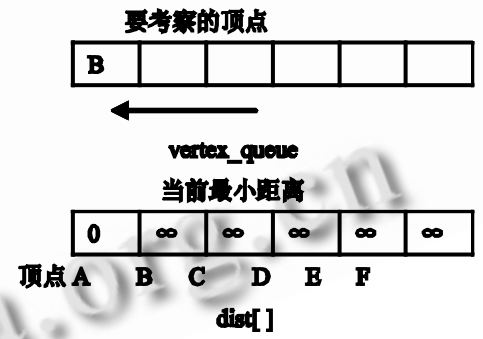
3　两种算法在图的搜阶段的并行性分析[4,5]

现仍用爬山图作为例子，按照搜索过程逐步进行分析：

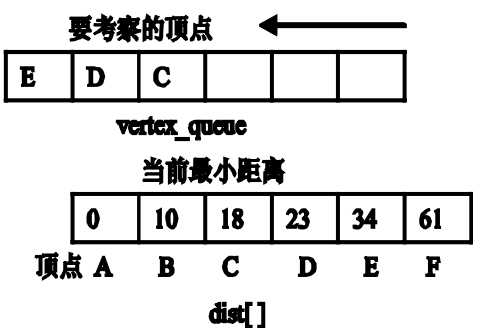
顶点和当前最小距离这两个关数据结构的初值为：

当 A 为源顶点时，元素 dist[A]总是零。如果把 A 选作源点，则该结构就提供了完整的通用性。

(1) 首先，检查从顶点 A 发出的每条边，在我们的图中，将是顶点 B 。到 B 的权值为 10，它提供了到 B 的第一个距离(实际上是唯一的距离)。两个数据结构 vertex_queue 和 dist[]更新如下：



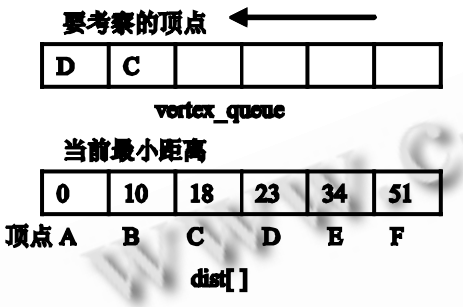
(2) 一旦一个新顶点 B 放入顶点队列，围绕 B 的搜索任务就开始了。现在还有四个边要检查：即到 C、D、E、F。这时在 Moore 算法中，不必以特定的次序检查这些边。但 Dijkstra 算法却要求首先检查最近的顶点，从而必须按顺序处理的方式进行处理。



我们现在就以 F、E、D、和 C 的次序来检查各边。经过顶点 B 到各顶点的距离分别为 dist[F]=10+51=61，dist[E]=10+24=34，dist[D]=10+13=23，dist[C]=10+8=18。因它们都是新距离，故所有顶点都要加到队列中(F 除外)：

顶点 F 之所以不需要加入，是因为它是终点、没有出边且不需处理(如果加入 F，则会发现没有出边)。

(3) 从 E开始,它有一个权值为17的边到顶点 F。通过顶点 E 到 F 的距离是 dist[E]+17=51，它比当前到 F 的距离小，故要替换这一距离，导致：



(4) 接下来是顶点 D，它有一条边到顶点 E，权值为 9,经过 D到 E的距离为 dist[D+9]= 23+9=32，它当前到 E 的距离小，故要替换这一距离，把顶点 E 加到队列，如下：

(5) 接下来是顶点 C，它有一条边到顶点 D，权值为 14，因此，经过 C 到 D 的距离为 dist[C]+14=18+14=32，它当前到 D 的距离 23 大，该距离保持不变。

(6) 接下来是顶点 E(又一次)，它有一条边到顶点 F，权值为 17，使经过 E 到 F 的距离为 dist[E]+17=32+17=49，它比当前到 F 的距离小，故要替换这一距离，如下。

现在再没有顶点要考察。我们得出从顶点 A 到其他各点的最短距离，包括终点 F。通常除了距离外，还要实际路径，那么在记录距离时就需把路径存储下来，上述中的路径为：A→B→D→E→F。

综上所述，在图搜索的第二阶段中，若使用 Dijkstra 算法则要求首先检查最近的顶点，因而 Dijkstra 算法在这个阶段中进行图的搜索时则必须按照顺序处理的方式进行搜索。这样 Dijkstra 算法在图的搜索阶段的并行性能也就大大降低了。这无形中给多处理机系统的应用带来了极大的负面影响。故我们只能选择 Moore 算法。

4　Moore算法在多处理机系统中的高效并行实现方案

为了使 Moore 算法在多处理系统中能够高效地实现，我们采用了集中式工作池和分布式工作池方式两种解决方案来提高其并行性。

4.1 在集中式工作池中的高效并行实现

利用集中式工作池来提高 Moore 算法在多处理机系统中的并行性能时，首先将顶点队列 Vertex_queue[]作为任务存放在集中式工作池中,每个从进程从顶点队列取得顶点，按照图 5 所示的方式返回新的顶点。对于要确认边和计算距离的从进程，它们需要访问存放图权值的(邻接矩阵或邻接列表)数组和存放当前最小距离的数组(dist[])。如果这些信息由主进程持有，它们就要向主进程发送消息以访问这些消息，这会导致非常大的通信开销。由于存放图权值的结构

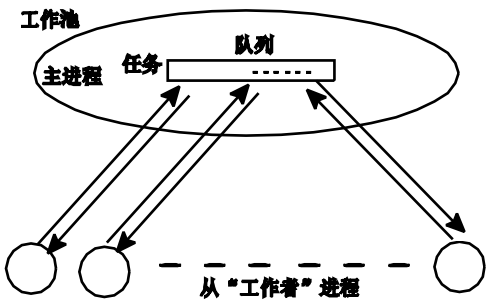


图 5　集中式工作池

是固定的，可将该结构拷贝到每个从进程中，假定使用的是拷贝的邻接矩阵。现在，我们再假设距离数组dist[]为中央存放的，且与顶点一起整体拷贝，也可单独对距离进行请求。代码可为如下形式：

主进程：

```
while(vertex_queue() != empty) {
    recv(PANY,Source-=Pi); /* 从进程任务请求*/
    v=get_vertex_queue( );
    send(&v,Pi);              /*发送下个顶点和 * /
    send(&dist,&n,Pi);        /* 和当前距离数组dist[] */
    send(&j,&dist[j],PAny,source=Pi);    /* 新的距离 */ append_queue(j,dist[j]); }; /*增加队列顶点和更新距离数组*/
    recv(PANY,source=Pi);  /*从进程任务请求*/
    send(Pi,termination_tag);    /*消息终止*/
    从进程(进程 i)
    send(Pmaster);            /*发送请求任务*/
    recv(&v,Pmaster,tag);        /*获取顶点数*/
    if(tag !=termination_tag) {
    recv(&dist,&n,Pmaster);      /*获取距离数组dist[] */
    for (j = 0;j<n;j++)          /*获取下一条边 */
    if (w[v][j] != infinity) {   /*如果该条边存在*/
        newdist_j = dist[v] + w [v][j];
        if(newdist_j<dist[j]) {dist [j] = newdist_j;
send(&j,&dist[j],Pmaster);}}}  /*添加顶点队列发送更新的距离*/
```

明显，顶点数和距离数组可放在一个消息中发送。还要注意各个从进程或许有不完全一样的距离，因为它们是由不同的从进程连续更新的。

4.2 在分布式工作池中的并行实现

实际应用中还可以用分布式工作池方法来求解问题。任务队列，下面的从进程代码中的 vertex_queue []也可以是分布式的。有一种方便的方法是：让从进程 i 只围绕顶点 i 搜索，如果顶点 i 在队列中存在，就让进程 i 拥有顶点 i 的队列项。换句话说，队列中有一个元素专门用来存放顶点 i，该项在进程 i 中。数组 dist[] 也分布在进程中间，以便进程 i 保存当前到顶点 i 的最小距离，为了确认顶点 i 的边，进程 i 还需存储顶点 i 的邻接矩阵/列表。

根据我们的安排，算法可按如下方式进行。由一协调进程激活搜索，将源顶点装载到适当的进程。在我们的例子中，A 是第一个搜索的顶点。首先激活指派给 A 的进程，该进程立即在顶点周围开始搜索，找出到相连顶点的距离；然后，把该距离送到相应的进程。到顶点 j 的距离将被送到进程 j，与它当前存储的值相比较，如果当前存储的值较大就被替换。在例中，到顶点 B 的距离与负责顶点 B 的进程联系。按这种方式，在搜索中将更新所有的最小距离，如果 d[i]的内容改变，进程 i 就要被重新激活，再次搜索。图 6 显示了消息传递的情况，注意消息传递分布在许多从进程间，而不是集中在主进程上。

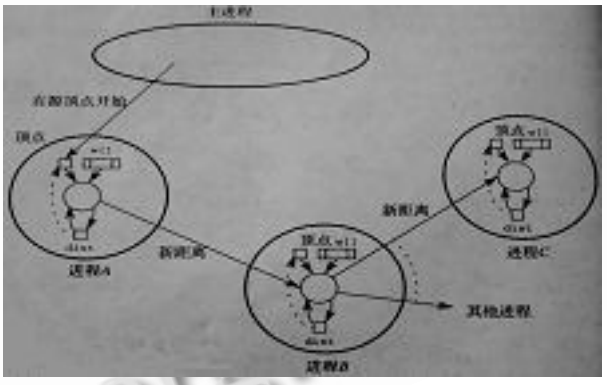


图 6　分布式图搜索

从进程的代码段可为如下形式：

```
从进程(进程 I)
recv(newdist,PANY);
if (newdist<dist)  { dist=newdist;
vertex_queue=TRUE;          /*加到队列中*/
} else vertex_queue ==PALSE;
if (vertex_queue==TRUE)     /*开始搜索周围的顶点*/
    for (j=0; j<n;  j++)            /*获取下一条边*/
        if (w[j] !=infinity)  { d=dist+w[j];
```

(下转第 124 页)

(上接第 80 页)

```
send(&d,Pj); }      /*传送距离到进程 j 中*/
```

上述代码可简化为：

```
从进程(进程 i)
recv(newdist,PANY);
if (newdist<dist) {dist=newdist; /*开始搜索周围的顶点*/
for(j=1;j<n,j++)      /*获取下一条边*/
if(w[j] !=infinity) { d=dist+w[j];
send(&d,Pj); }      /*传送距离到进程 j 中*/
```

如上所述，我们还需要用一种机制来重复这些动作，并在所有进程空闲时终止，该机制必须处理传输中的消息。实现这一机制的有效而简单的解决方法是：利用同步消息传递，其中一个进程只有在对方收到消息后才能继续运行。

值得我们注意的是，一个进程只有在其顶点放入顶点队列之后才是活动的。有可能很多进程不是活动的，从而使这种解决方案变成低效的。如果将一个顶点分到每个处理器，则该方法对大图也是不实用的，在那种情况下，我们可以把一组顶点分配到一个处理器上以提高该方案的适用范围。

5 结语

通过实验我们发现：图搜索的第二阶段由于 Dijkstra 算法要求首先检查最近的顶点，从而必须按顺序处理的方式进行处理，而 Moore 算法则不必以特定的次序检查各个边可以进行并行处理；在实验中我们又采用了集中式工作池和分布式工作池工作池两种方式来实现各处理器之间动态负载平衡，极大地提高了多处理机系统的工作效率，充分利用 CPU 资源。达到我们预期的目的。

参考文献

1 杜端甫,编.运筹图论.北京:北京航空航天大学出版社, 1990.

2 严寒冰,刘迎春.基于 GIS 的城市道路最短路径算法探讨.计算机学报, 2000,23:200－215.

3 Moore EF, The shortest path through a maze.Proc. Int. Symp. on Theory of Switching Circuits,Part II, April 2-5, 1957,4:285－292.

4 谢政,李建平.网络算法与复杂性理论.长沙:国防科技大学出版社, 1995.

5 康立山.非数值并行算法(第 1 册).北京:科学出版社, 2003.