文章编号:1001 - 9081(2005)12 - 2921 - 02

一种实用的所有点对之间最短路径并行算法

周益民,孙世新,田 玲
(电子科技大学 计算机科学与工程学院,四川 成都 610054)
(ehearsay@163.com)

摘 要:针对有向图中每对顶点之间的最短路径问题,在基于扩充了路径矩阵的串行 Floyd 算法上,提出了二维网格结构上的并行算法。选用的任务划分方法为二维均匀块分配方法。该并行算法已经在 NOW 上的 MPI 平台上实现,理论分析和数值实验表明它具有较高的扩展性和并行效率。

关键词:所有点对之间最短路径;Floyd 算法;并行算法
中图分类号: TP301.6 **文献标识码**:A

Practical parallel algorithm for all-pairs shortest-path problem

ZHOU Yi-ming, SUN Shi-xin, TIAN Ling
(*College of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*)

Abstract: Aiming at the all-pairs shortest-path problem in the directed graph, a practical parallel algorithm, which based on the Floyd algorithm with an extended path array, was brought forward on 2-D mesh network . The planar evenly partition method was chosen for task division in this parallel algorithm. The parallel algorithm was implemented on MPI on NOW. The theoretical analysis and the experimental results prove that the parallel algorithm is an efficient and scalable algorithm.

Key words: all-pairs shortest-path problem; Floyd algorithm; parallel algorithm

0 引言

随着计算机的出现和发展,图论的研究得到广泛重视,最短路径问题作为图论中的一个典范问题,它已经被应用于众多领域。在网络通信领域,信息包传递的路径选择问题也与最短路径问题息息相关。解决最短路径问题的算法在诸多工程领域都有较强的实用价值。一般来说最短路径问题分为单源最短路径问题和全源最短路径,公认的比较好的算法 Dijkstra 算法比较适合于单源最短路径,而 Floyd 算法适合用于求解全源最短路径。

本文所关注的所有点对之间最短路径问题也就是全源最短路径问题,就是要求解一个有向带权图中所有顶点之间的最短路径,因此选用 Floyd 算法作为算法基础。

以 Floyd 算法作为基础的并行算法由于其划分方式的不同,有多种算法形式。文献[2]中给出了基于一维划分方法的 Floyd 并行算法,文献[1]中描述了基于一维和二维划分方法的 Floyd 并行算法,这些给出的求解最短路径的 Floyd 算法研究都偏重于求最短路径长度,有一个共同的不足之处:即不能高效率地求出一个顶点到另一个顶点的最短路径具体经过的顶点序列。因此在求解所有点对之间最短路径问题时出现了 F-D 算法[3]。文献[4]中给出了在 Floyd 串行算法基础上扩展了路径矩阵的一个串行算法,提供了一种比较便利的求解最短路径问题时顶点序列跟踪和记录的方法,而且整个串行算法的时间复杂度仍然保持在 $\Theta(n^3)$。

本文在扩充了路径矩阵的 Floyd 串行算法[4]基础上,进行了并行化设计,提出了一个在二维网格结构上求解所有顶点之间的最短路径的并行算法。数值实验和分析说明,该并行算法具有实现简单、扩展性强、适用于大规模计算等特点,有良好的性能和实用价值。

1 并行算法的设计

1.1 扩充了路径矩阵的 Floyd 串行算法

Floyd 算法,又称传递闭包方法,其实质就是邻接矩阵自乘 n 次,算法的时间复杂度为 $\Theta(n^3)$[1]。下面就是扩充了路径矩阵的 Floyd 串行算法的描述:

算法 1:Floyd 串行算法

输入:顶点数目 n;图的邻接矩阵 ***array***;初始化后的路径矩阵 ***path***

```
计算:for (k = 0;k < n;k ++)
       for (i = 0;i < n;i ++)
         for (j = 0;j < n;j ++)
           if (array[i][j] > (array[i][k] + array[k][j]))      ①
             {
               array[i][j] = array[i][k] + array[k][j];        ②
               path[i][j] = path[i][k];                        ③
             }
```

输出:变换后的矩阵 ***array*** 和 ***path***

算法 1 中,邻接矩阵是一个 $n \times n$ 的矩阵 ***array***,在本文中用 ∞ 来表示不存在的边,但在具体的实现中用非常大的值(如所能表示的最大整数)来表示不存在的边。为了表示求得的任意两个顶点最短路径具体经过途径中各个顶点的顺序,构造路径矩阵 ***path***,来实现对最短路径途径中各个顶点的追踪和记录。顶点路径矩阵 ***path***$[n][n]$ 的初始化为 ***path***$[i][j] = j(0 \leqslant i,j \leqslant n-1)$。对于大规模的图求解所有点对之间最短路径,Floyd 算法的串行算法 $\Theta(n^3)$ 这样的时间复杂度要花费较大的运算时间。因此,通过对串行算法的并行化改造来缩短计算时间提高效率是很有必要的。

收稿日期:2005 - 09 - 02

作者简介:周益民(1980 -),男,四川邛崃人,博士研究生,主要研究方向:并行算法、大规模并行计算; 孙世新(1940 -),男,湖北孝感人,教授,博士生导师,主要研究方向:并行算法及其应用、网格计算; 田玲(1981 -),女,四川成都人,硕士研究生,主要研究方向:网络及并行计算.

1.2 并行算法任务的划分方法

在并行算法的设计中,问题的分解通常可以有两种形式:一种是域分解,即将问题分解为若干个较小的问题区域,然后分别对其求解;另一种是功能分解,即将问题按功能分解为若干个子问题,而后对各个子问题并行求解。对于所有点对之间最短路径问题,在采用了 Floyd 的算法思想后,选择是很明显的。通过算法 1 的分析可以看出,算法对同一组判断和赋值语句①②③执行了 n^3 次。除非可以再进一步分割这些语句,否则无法找出功能分解来达到功能并行。但是可以很容易地进行域分解:把输入的矩阵 ***array*** 和构造的矩阵 ***path*** 按块分解成任务,并对每一个块的任务数据元素执行操作。

本文选择的任务划分方法为二维均匀块分配方法。这种均匀块分配方法在计算中可以使各个任务的并行计算中达到负载平衡。而且这种划分方法特别适合于拓扑结构为二维网格型的处理器网络,在实际应用中具有很强的扩展性。

1.3 任务之间的数据通信

通过对 Floyd 串行算法的分析,可以获得并行算法中任务之间通信所需要的数据的选取。

从算法 1 中的①②③可以看出来,更新 ***array***[i][j] 仅仅需要访问元素 ***array***[i][k] 和 ***array***[k][j]。对于给定的 k,所有元素的更新除了需要其自身之外,也仅仅是需要第 k 行和第 k 列中的相应元素。同样的道理也适用于路径矩阵 ***path***,更新 ***path***[i][j] 仅仅需要访问元素 ***path***[i][k],在第 k 层的循环中也只需要 ***path*** 矩阵的第 k 列元素。

又因为在第 k 层的循环迭代中,对 ***array***[i][k] 和 ***array***[k][j] 的更新是按照算法 1 中的 ① 的方式进行的:

if(*array*[i][k] > (*array*[i][k] + *array*[k][k]))

if(*array*[k][j] > (*array*[k][k] + *array*[k][j]))

两条判断语句很明显不可能为真,那么在第 k 层的迭代中 ***array***[i][k]、***array***[k][j] 和 ***path***[i][k] 的值都不会执行算法 1 中的 ② 和 ③,那么它们的值不会在第 k 层改变。因此,对于外层的第 k 次循环来说,可以先广播第 $k-1$ 层的循环中 ***array*** 的第 k 行和第 k 列元素、***path*** 的第 k 列元素,然后各个任务并行地进行运算。

图 1 描述的一个输入矩阵 ***array*** 和构造矩阵 ***path*** 都是 $n \times n = 16 \times 16$ 阶的矩阵,处理机 $p = 16 = 4 \times 4$,把两个矩阵都按照图中的方法划分任务,每台处理机都负责矩阵 ***array*** 和矩阵 ***path*** 的一块,本例中有 16 块数据分割到了 16 个处理机,每台处理机负责 4 行 4 列数据。

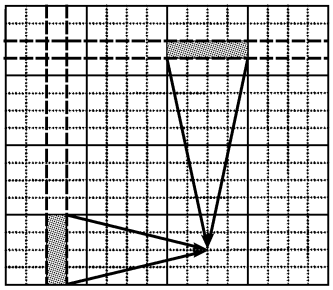

图 1 当 $k = 3$ 时,第 14 号处理机在接受数据

在二维划分的模式中可以观察到,更新各个处理机的任务数据元素其实只需要 ***array*** 矩阵的第 k 行中的长度为 $n/\sqrt{p}$ 的一段。同理,也只需要 ***array*** 矩阵第 k 列的长度为 $n/\sqrt{p}$ 的一段和 ***path*** 矩阵第 k 列的长度为 $n/\sqrt{p}$ 的一段。图 1 是当 $k = 3$ 时,第 14 号处理机需要的元素的例子。

1.4 并行算法的实现

为了方便起见,约定处理机数为 $p = \sqrt{p} \times \sqrt{p}$,数据规模 n,$\sqrt{p}$ 可以整除 n。如果有 p 台处理机,那么就划分 p 个任务,也保证 p 台处理机所负担的任务数据量相同。

下面是二维均匀块分配方法分解任务的 Floyd 并行算法。

算法 2:二维任务划分方法的 Floyd 并行算法

输入:顶点数目 n;图的邻接矩阵 ***array***;初始化后的路径矩阵 ***path*** 到主机(0 号进程)

计算:

(1) 如果是主机(0 号进程),根据数据规模 n 和处理机数目 p 来划分任务,按照棋盘式均匀的分解的方法,发送 ***array*** 和 ***path*** 的数据块到相应的处理机,各处理机在本地存储为 ***jobarray*** 和 ***jobpath***。

(2) for ($k = 0$;$k < n$;k ++) 循环开始。

(3) 各个处理器计算当前需要广播的第 k 行 ***array*** 所在的处理机号。如果在本地,那么从 ***jobarray*** 矩阵中选取出相应的一行数据,存储在 *temparrayA*[ ],并向同一列处理机广播。各个处理器计算当前需要广播的第 k 列 ***array*** 和 ***path*** 所在的处理机号。如果在本地,那么从 ***jobarray*** 和 ***jobpath*** 矩阵中选取出相应的一列数据,分别存储在 *temparrayB*[ ] 和 *temppath*[ ],并向同一行处理机广播。

(4) 各个处理器执行并行计算:

for ($i = 0$;$i < n/\sqrt{p}$;i ++)

for ($j = 0$;$j < n/\sqrt{p}$;j ++)

if (***jobarray***[i][j] > (*temparrayB*[i] + *temparrayA*[j]))

{***jobarray***[i][j] = *temparrayB*[i] + *temparrayA*[j];

path[i][j] = *temppath*[i];}

(5) 跳(2) 直至 k 的循环结束转(6);

(6) 各个处理器返回运算结果 ***jobarray*** 和 ***jobpath*** 两个矩阵到主机(0 号进程),主机在将其归并到本地的 ***array*** 和 ***path*** 矩阵。释放空间。

输出:变换后的矩阵 ***array*** 和 ***path***。

1.5 并行算法复杂度分析

算法 2 中的第(4)步是用来更新矩阵 ***array*** 和 ***path*** 的判断和赋值语句,时间复杂度为 $\Theta((n/\sqrt{p}) \times (n/\sqrt{p})) = \Theta(n^2/p)$;在中间循环,也就是第(3) 步,从一个处理器构造三个长度为 $n/\sqrt{p}$ 的消息到另外一个处理器的时间复杂度为 $\Theta(3n/\sqrt{p})$。由于广播到 $\sqrt{p}$ 个处理器需要 $\log\sqrt{p}$ 个消息传递步骤,所以每个循环迭代的广播操作总的时间复杂度为 $\Theta(3n\log\sqrt{p}/\sqrt{p})$。并行算法的最外层循环,也就是第(2) 步,执行了 n 次迭代,因此并行算法总的时间复杂度为:

$$\Theta\left(n \times \frac{n^2}{p}\right) + \Theta\left(3 \times \frac{n\log\sqrt{p}}{\sqrt{p}}\right) = \Theta\left(\frac{n^3}{p}\right) + \Theta\left(\frac{n^2\log p}{\sqrt{p}}\right)。$$

根据串行运行时间复杂度为 $\Theta(n^3)$,得到加速比和效率如下:

$$S = \frac{T_1}{T_p} = \frac{\Theta(n^3)}{\Theta\left(\frac{n^3}{p}\right) + \Theta\left(\frac{n^2\log p}{\sqrt{p}}\right)}$$

$$E = \frac{1}{1 + T_o/T_1} = \frac{1}{1 + \Theta\left(\frac{n^2\log p}{\sqrt{p}}\right)\Big/\Theta\left(\frac{n^3}{p}\right)}$$

$$= \frac{1}{1 + \Theta\left(\frac{\sqrt{p}\log p}{n}\right)}$$

(下转第 2934 页)

识别成障碍物或非道路区域,这是由于仅用颜色作为判断标准的结果,对此还需要进一步研究以克服这种局限性,从而提高道路区域的完整性和道路检测的正确率。

参考文献:

[1] BROGGI A. Robust Real-Time Lane and Road Detection in Critical Shadow Conditions[EB\OL]. http://www.ce.unipr.it/people/broggi/publications/coralgables.pdf, 1995.

[2] HANDMANN U, KALINKE T, TZOMAKAS C, *et al*. An image processing system for driver assistance[EB\OL]. http://citeseer.ist.psu.edu/handmann98image.html, 1998.

[3] GONZALEZ RC, WOODS RE. Digital Image Processing Second Edition[M]. Prentice Hall, 2002.

[4] 张燏, 吴志斌, 陈淑珍, 等. 一种新的自适应二值化方法[J]. 计算机工程, 2002, 28(5): 184-186.

[5] JEONG H, OH Y, PARK JH, *et al*. Vision-based adaptive and recursive tracking of unpaved roads[J]. Pattern Recognition Letters, 2002, 23(2-3): 73-82.

[6] 李青, 郑南宁, 马琳, 等. 基于主元神经网络的非结构化道路跟踪[J]. 机器人, 2005, 27(5): 247-251.

[7] 苏开娜, 任文君, 易小琳, 等. 基于运动模型的道路识别与跟踪算法的研究[J]. 中国图象图形学报, 2000, 5(3): 226-230.

[8] 邢征北, 郑南宁, 刘铁, 等. 道路检测算法及其 DSP 实现[J]. 微电子学与计算机, 2004, 21(5): 114-117.

[9] DESOUZA GN, KAK AC. Vision for mobile robot navigation: a survey[J]. IEEE Transaction on Pattern and Analysis and Machine Intelligence, 2002, 24(2): 237-267.

[10] GREGOR R, LUTZELER M, PELLKOFER M. EMS-Vision: A Perceptual System for Autonomous Vehicles[J]. IEEE transactions on intelligent transportation systems, 2002, 3(1).

[11] THORPE C, JOCHEM T, POMERLEAN DA. Automated highways and the free Agent demonstration[A]. Proceedings of Int Symp Robotics Research[C]. 1997.

(上接第 2922 页)

其中,T_1 表示串行算法在单处理机上的运行时间,T_p 表示并行算法在 p 台处理机上的运行时间,T_o 表示并行算法在 p 台处理机上执行时的通信、同步等时间开销。

对于成本最优的并行形式,$\frac{\sqrt{p}\log p}{n} = O(1)$,则二维划分 Floyd 并行算法可以有效地使用最多 $O\left(\frac{n^2}{\log^2 n}\right)$ 个进程,获得整体的等效率函数为 $\Theta(p^{1.5}\log^3 p)$。

2 数值实验结果和分析

在由四台微机(联想 PIII800MHz)组成的集群系统上进行了数值实验,并行程序用VC++编写的,并在 MPI 环境下运行后得出最终实验数据。表 1 为并行算法在各种数据规模上的运行时间和加速比。

表 1 并行算法在各种数据规模上的运行时间与加速比

矩阵规模 n	$p=1$	$p=4$		$p=16$(模拟)	
	时间/s	时间/s	加速比	时间/s	加速比
200	0.35	0.72	0.481		
300	1.52	1.53	0.994		
400	3.58	3.45	1.461	1.393	2.57
500	6.99	3.74	1.869		
600	12.0	5.35	2.250		
700	19.03	7.116	2.674		
800	28.38	10.07	2.818	4.370	6.494
900	40.29	13.15	3.064		
1000	55.26	17.41	3.174		
1100	73.50	22.58	3.260		
1200	95.48	29.23	3.267	9.889	9.655
1300	121.2	36.48	3.323		
1400	151.3	44.46	3.39		
1500	186.9	55.48	3.35		
1600	226.7	66.55	3.39	19.86	11.4

图 2 为并行算法的效率图。$p=4$ 时,在数据规模小于 1000 的阶段,并行效率由低到高增长明显。这是因为在数据规模比较小的时候,通信延迟占用了大量的时间开销,随着数据量的增加,处理器计算时间和数据通信时间比值也随之增大,算法中大量的时间都用于计算,所以效率逐渐增大。当规模大于1000后,效率增长趋于平缓,但效率达到了80%以上,获得了比较高的并行效率。

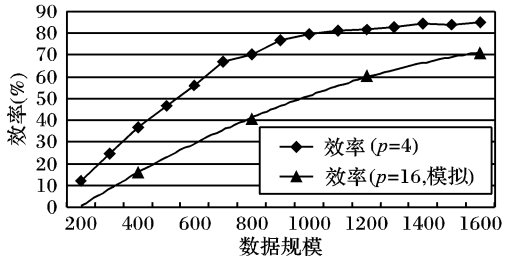

图 2 并行算法的效率

在相同的数据规模下,$p=16$ 时的并行效率明显要低于 $p=4$ 时的并行算法,这是因为当 $p=16$ 时,分割任务为 16 块,数据的通信开销要比 $p=4$ 时候大为增加。在模拟的环境下,数据规模达到1600 的时候,$p=16$ 的并行算法效率达到了 70%,在实际应用中也是可以接受的。

3 结语

针对所有点对之间最短路径问题的 Floyd 扩充路径并行算法,本文提出一种基于二维网格结构的并行算法。该并行算法实现简单,能够处理不同规模的数据并且在大规模并行计算上有着广泛的应用前景。

实验结果表明,该并行算法降低了计算所有点对之间最短路径问题的计算时间,提高了计算效率,具有良好的并行加速比。从扩展性来看,它可以适用于 NOW 系统,也适用于各种规模的二维处理器网格型系统。

算法的不足之处在于最后获得的路径结果矩阵 ***path*** 还需要进一步的整理才可以获得求得的最短路径具体经过的顶点序列,这也是今后的工作所在。

参考文献:

[1] GRAMA A, GUPTA A, KARYPIS G, *et al*. Introduction to Parallel Computing[M]. Second Edition. Harlow England: Addison-Wesley, 2003. 429-467.

[2] QUINN MJ. MPI 与 OpenMP 并行程序设计: C 语言版[M]. 陈文光, 武永卫, 等译. 北京: 清华大学出版社, 2004. 111-125.

[3] 程晓荣, 刘斌, 陆旭, 等. F-D 算法求解最短路径[J]. 华北电力大学学报. 2003, 30(6): 75-77.

[4] 周炳生. Floyd 算法的一个通用程序及在图论中的应用[J]. 杭州应用工程技术学院学报. 1999, 11(3): 1-9.

[5] FLOYD RW. Algorithm 97: Shortest path[J]. Communications of the ACM, 1962, 5(6): 345.