基于最短路径的道路网络k近邻查询处理

廖 巍[1] 吴晓平[1] 胡 卫[1] 钟志农[2]
(海军工程大学电子工程学院 武汉 430033)[1] (国防科技大学电子科学与工程学院 长沙 410073)[2]

摘 要 针对基于空间道路网络的k近邻查询处理,提出了分布式移动对象更新策略以有效减少服务器计算代价,利用基于内存的空间道路网络邻接矩阵、最短路径矩阵结构和移动对象哈希表索引分别对道路网络无向图与移动对象进行存储管理。提出了基于最短路径度量的网络扩展搜索(SPNE)算法,以通过裁剪网络搜索空间来减少k近邻查询搜索代价。实验表明,SPNE算法的性能优于传统的NE和MKNN等k近邻查询处理算法。

关键词 空间道路网络,k近邻查询,最短路径矩阵,SPNE算法
中图法分类号 TP392 **文献标识码** A

Processing of k Nearest Neighbor Queries Based on Shortest Path in Road Networks

LIAO Wei[1] WU Xiao-ping[1] HU Wei[1] ZHONG Zhi-nong[2]
(Department of Electronic Engineering, Naval University of Engineering, Wuhan 430033, China)[1]
(Department of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)[2]

Abstract To efficiently process k nearest neighbor queries in spatial road networks, this paper presented a distributed moving objects updating strategy to reduce the computing cost of server. In-memory spatial network adjacent matrix, shortest path matrix and hash table structures were introduced to describe the road network topology and store the moving objects. A shortest path based network expansion (SPNE) algorithm was proposed to decrease the processing cost of k nearest neighbor queries by reducing the search network space. Experimental results show that the SPNE algorithm outperforms existing algorithms including NE and MKNN algorithms.

Keywords Spatial road networks, k-NN queries, Shortest path matrix, SPNE algorithm

1 引言

随着近年来无线通信、GPS空间定位技术的快速发展,以及具有定位功能的无线手持和车载设备的大量普及,位置服务(Location-based Service)目前已经得到了人们越来越多的关注。位置服务在地图导航服务、救援服务、战场指挥等领域有着广泛的应用前景[1]。尤其是在城市道路网络环境下,许多新型应用如交通导航、位置感知广告服务等由于切合用户的实际需求,引起了人们广泛关注。在线导航服务Google Maps和Microsoft MapPoint等目前已能够提供简单的空间定位和路径规划服务[2],如查询从当前位置到目的地的最短路径等。随着用户规模的不断扩大,并且考虑到用户位置具有不断变化的动态性,以在线导航服务为代表的位置服务系统必须能够高效及时地响应用户查询请求,因此对查询处理算法的性能方面提出了很高的要求。比如移动用户A提出查询请求"查找距离我当前位置最近的k个出租车",由于用户A和出租车的位置都可能随时发生变化,过久的查询响应时间延迟将会导致当前查询结果变为无效。因此,高效的空间道路网络查询处理技术目前已成为地理信息系统和空间数据库领域的研究热点之一。

针对基于道路网络的空间查询,Papadias[3]首先提出了基于空间网络的查询处理框架,通过在空间网络中集成欧几何(Euclidean)信息,采用网络扩展(network expansion)策略来对网络边依次扩展搜索进行查询处理。Kolahdouzan[4]提出的VN^3技术将预计算的空间Voronoi图与R树索引混合存储,通过查找Voronoi图来获得k近邻对象,从而避免在线计算代价。Cho等人[5]基于Dijkstra算法提出了UNICONS技术,针对预知查询点运动轨迹的k近邻查询处理,将预计算的k近邻对象存储在最短路径节点,通过查询搜索即可获得k近邻对象。Wang等人[6]提出的MKNN算法考虑到道路网络的拓扑稳定性和移动对象的动态性,利用磁盘R树结构和内存格网索引结构对道路网络和移动对象分别进行存储,以减少k近邻查询处理的磁盘I/O代价。

目前看来,现有技术大都采用基于磁盘和内存的混合索引结构,在较小的内存占用空间前提下研究高效的查询处理算法。随着硬件技术的快速发展,内存容量不断增大,64位处理器逐渐成为主流,极大地增加了计算机内存的可用容量。因此,对于空间网络查询这种查询和更新密集型应用而言,将所有数据放入内存进行处理以提高查询性能成为现实[7]。基于此,本文针对基于空间道路网络的k近邻查询,考虑到移动终端计算能力提出了分布式移动对象更新策略,通过引入基于内存的空间道路网络邻接矩阵、最短路径矩阵结构和移动

到稿日期:2009-12-02 返修日期:2010-02-26 本文受国家863高技术发展计划(2007AA12Z208),中国博士后科学基金(20080431384)资助。
廖 巍(1980—),男,博士后,讲师,主要研究方向为空间数据库等,E-mail:liaowei_2000@163.com。

对象哈希表索引结构，在空间道路网络拓扑中集成最短路径信息，提出了基于最短路径长度为裁剪度量准则的网络扩展搜索(SPNE)算法，并以实验对比和分析验证了SPNE算法的有效性。

2 问题描述

在许多实际应用如地图导航服务系统中，道路网络通常表示为由大量的道路节点和道路边构成的集合。实际道路网络通常被映射为内存中的有向图或无向图模型来进行空间查询处理[6,8]。本文将实际道路网络抽象为无向图模型进行表示，最短路径长度为无向网络距离，针对有向图道路网络模型，将所提出的索引结构和查询处理算法只要稍作扩充即可适用。具体而言，无向图的顶点表示为道路节点的集合；无向图的边表示为路段的集合；权值可以表示为该路段的长度、拥塞程度等；道路网络中的移动对象表示为图边上的运动点。本文假定道路网络特性为非易变的，即道路网络对应的无向图拓扑结构和网络边的权重始终不会发生改变；显然可知，道路网络中任意两个顶点之间的最短路径始终保持不变。

定义1 一个(道路)网络可以表示为无向带权图 $G=(V,E)$。其中，V 是顶点(如交叉点、终点等)的集合，E 是边(路段)的集合，而且有 $E\subseteq V\times V$。

定义2 在(道路)网络 $G=(V,E)$ 中，每条边 $e(s,t)\in E$，表示该边由顶点 s 和 t 连接，每条边 e 都赋有权值，表示为函数 $W:E\rightarrow R^+$，其中 R^+ 为正的实数集合。

我们用 M 表示道路网络边(路段)上运动的移动对象(包括行人、车辆等)集合，移动对象 $o\in M$ 的当前位置表示为 $loc(o)=(x,y)$。查询 q 是 M 中的移动对象在当前时刻发出的空间查询。本文主要研究基于当前位置的k近邻查询(如“查找距离我当前位置最近的出租车”等)，以道路网络距离为空间度量。

定义3 两个移动对象之间的当前距离表示为 $dist(o_1,o_2):loc(o_1)\times loc(o_2)\rightarrow R^+$。$dist(o_1,o_2)$ 表示当前时刻在网络度量空间中 o_1 到 o_2 的最短路径长度。

定义4 查询点 q 到网络边 e 的当前最小距离 $dist(q,e)$ 表示为查询点 q 到 e 的两个顶点 s 和 t 的最短路径长度的最小值，即 $dist(o_1,o_2)=\min(dist(q,s),dist(q,t))$。

基于空间道路网络的k近邻查询含义是指查找空间道路上距离查询点 q 最近的k个移动对象。由于移动对象在空间道路网络上可任意运动，空间位置会随着时间而不断变化，因此，距离查询点 q 距离最近的k近邻对象也会随着空间道路网络上移动对象位置的改变而发生变化。目前国内外大部分工作都是研究空间道路网络的移动对象的当前位置k近邻查询，即根据空间道路网络上运动的移动对象的当前位置快照信息，计算出当前时刻距离查询点 q 最近的k个移动对象。考虑到查询结果具有很强的时效性，现有研究通常采用周期性的更新策略对查询结果进行定时更新，用户可以通过设定更新周期间隔来获得满意的准确性。

3 内存索引结构

基于空间道路网络的k近邻查询采用网络距离度量，与欧几何空间k近邻查询不同，服务器在处理用户提交的k近邻查询时，在获得移动对象的当前位置信息后必须通过搜索匹配，将该移动对象映射关联到相应的道路网络边上，然后基于道路网络拓扑图来进行搜索处理。考虑到目前移动终端通常具备一定的计算和存储能力，除了能够利用GPS等定位方式获得其空间坐标外，还能够根据当前空间位置信息与道路网络地图信息计算出所在的道路网络边。因此，本文提出了基于移动终端的分布式移动对象更新策略，即当移动用户向服务器提交位置更新请求时，首先通过本地终端计算其所在的道路网络边，然后同时发送其当前位置、该移动对象标识以及所在道路边标识到服务器端，从而能够充分利用客户端计算资源来减轻服务器的更新负载。

本文采用 $N\times N$ 的邻接矩阵 A 来描述空间道路网络的拓扑结构，矩阵中元素 $A[i,j]\in A$ 表示道路网络节点 v_i 到 v_j 的最短路径所经由的起始道路网络节点，即从节点 v_i 到 v_j 的最短路径必须首先经过节点 $A[i,j]$。显然，若道路网络节点 v_i 和 v_j 相邻，那么有 $A[i,j]=v_j$，邻接矩阵 A 中包含了任意两个道路网络节点之间的邻接信息，因此通过搜索邻接矩阵 A 便可判断出道路节点之间是否邻接。另外，根据邻接矩阵 A，我们可以递归搜索查找任何两个空间道路网络节点之间的最短路径。空间道路网络拓扑的权重信息则采用 $N\times N$ 的最短路径长度矩阵 B 来进行描述，矩阵 B 中元素 $B[i,j]\in B$ 表示道路网络节点 v_i 到 v_j 的最短路径长度。显然，若道路网络节点 v_i 和 v_j 相邻，则 $B[i,j]$ 表示连接空间道路网络节点 v_i 和 v_j 之间的网络边权重。可以看出，矩阵 A 和 B 不仅能完备地描述空间网络的拓扑结构信息，而且包含了空间道路网络任意两个节点之间的最短路径信息。移动对象则利用基于内存的哈希表结构映射到不同的道路网络边上进行存储，其记录形式为元组 $\langle edgeId,o_1,o_2,\cdots,o_n\rangle$，其中 $edgeId$ 为道路网络边的唯一标识，$o_i(i=1,\cdots,n)$ 表示当前时刻落在该道路网络边上的移动对象标识，利用该结构便能够描述移动对象和道路网络边之间的位置映射关系。

4 基于最短路径的网络扩展(SPNE)搜索算法

基于上述内存索引结构，本文与文献[3]类似采用网络扩展搜索策略来处理基于空间道路网络的k近邻查询。与传统网络扩展搜索算法不同，本文提出的基于最短路径网络扩展(SPNE)搜索算法对网络边进行遍历搜索，在搜索过程中算法以k近邻距离为裁剪度量准则，根据查询点到该边的最短路径长度来对网络边进行裁剪，以减少搜索空间。SPNE算法具体思想如下：在周期性更新时刻，对用户提交的k近邻查询，算法首先获得查询点所在的网络边，查找该边上的移动对象以计算出初始k近邻距离；然后分别计算该查询点到与该边两个顶点邻接的所有网络节点的最短路径距离，并按照最短路径距离大小排序进行搜索，每次搜索完当前网络边上的移动对象则更新k近邻距离和候选移动对象结果集；若k近邻距离小于查询点到当前搜索网络节点的最短路径长度，则将该顶点从搜索顶点集合中标识为无效，否则，将当前网络节点的邻接节点添加到搜索网络节点集合。

算法1描述了基于最短路径的网络扩展(SPNE)算法的具体流程。首先，第1到3行，算法根据查询 q 当前位置和所在边标识，从移动对象哈希表结构中获得其所在网络边中的移动对象集合，查找k近邻对象候选集合并计算出初始k近邻距离；第4行，算法搜索与该边两个顶点相邻的网络节点，

将其按照查询 q 到该节点的最短路径长度大小将网络边压入堆栈 H；第 5 到 10 行，算法依次从堆栈中获取当前网络边，计算出查询 q 到当前边的最小距离 $dist(q,e)$，若 $dist(q,e)$ 大于当前 k 近邻距离 $q.kNN.dist$，则忽略该网络边并将其弹栈，同时在搜索节点集合 $q.tree$ 中将该边上到查询 q 最短路径的节点标识为无效；否则搜索该边上的所有移动对象，判断是否可能为 k 近邻对象，若是则将其加入到初始结果集中，重新计算出 k 近邻距离后将该边弹栈，同时算法将该边另一顶点作为候选搜索节点放入搜索节点集合 $q.tree$ 中。依次类推，直至堆栈 H 为空。

算法 1 基于最短路径的网络扩展(SPNE)算法

1. Initialize the empty heap H; set $q.kNN.dist=\infty$
2. Get the edge e that contains q, let e be the root of $q.tree$
3. Insert the m best objects in e into $q.result$; Update $q.kNN.dist$
4. Search the edges going from e; en-heap all edges into H according to its shortest distance from q
5. While $H\neq\phi$, get the top edge te in H
6. if $q.kNN.dist > dist(q,te)$
7. For each object o in te, Compute the distance between q and o
8. Update $q.result$, $q.kNN.dist$ and $q.tree$
9. En-heap all the edges going from te into H according its distance from q
10. Else skip te and set $te.s$ in $q.tree$ invalid, de-heap te
11. If the heap $H=\phi$, then return

5 实验与分析

本文使用基于道路网络的移动对象产生器[9]来生成模拟移动对象数据集。产生器输入为德国 Oldenburg 城市道路图，共有 6105 个道路节点和 7035 条道路边，输出为在道路网络上运动的移动对象集合，共包含 100k 个移动对象。移动对象用点坐标来表示，在起始时刻每个移动对象选择一个距离其最近的道路节点为目的地进行运动，移动对象到达目的地后则会重新随机选择一个新目的地和速度进行运动。在实验中我们将时间单元大小设置为 1 秒，即每隔 1 秒实验系统根据当前移动对象位置快照信息更新 k 近邻查询结果。随机选择一个移动对象用户发出 k 近邻查询请求，查询的近邻个数分别取 k=1，5，10，15，20，25。实验运行环境为 Pentium4 3GHz 的双核 CPU、512MB 内存、Windows XP 操作系统。实验系统首先读取 Oldenburg 城市道路图节点文件和道路边文件，构建基于内存的道路网络邻接矩阵、最短路径矩阵和哈希表索引结构，然后以随机生成的 100 次 k 近邻查询平均 CPU 计算时间代价来衡量算法性能。

分别比较 SPNE 算法与 NE 算法、MKNN 算法在不同条件下的查询与更新性能。图 1(a)所示为固定近邻个数 $k=10$，移动对象规模变化时对算法查询性能的影响。从图中可以看出，SPNE 算法查询性能随着移动对象规模的增大呈线性下降，但优于传统网络扩展搜索 NE 算法和 MKNN 算法，这是由于 SPNE 算法采用分布式更新策略和基于内存的哈希表结构，能够直接定位移动对象所在的网络边，且利用 k 近邻距离和最短路径长度度量对搜索空间进行了裁剪，因而查询性能优于传统网络扩展搜索 NE 算法，而 MKNN 算法采用基于磁盘和内存的混合索引结构，需要额外的磁盘 I/O 代价，故查询性能最差。图 1(b)所示为固定移动对象数据规模为 50k 时，k 近邻个数对查询性能的影响。从图中可以看出，SPNE 算法、MKNN 算法和 NE 算法的查询处理代价随着 k 个数的增加而呈线性增长，SPNE 算法具有最好的查询性能，而 NE 算法利用网络扩展搜索策略，性能优于采用混合索引结构的 MKNN 算法。

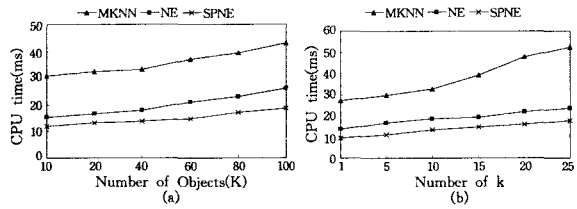

图 1 SPNE 算法查询性能比较

图 2(a)所示为固定近邻个数 $k=10$ 时，移动对象规模变化对查询更新性能的影响。从图中可以看出，SPNE 算法更新性能随着移动对象规模的增大呈线性下降，但优于传统 NE 算法和 MKNN 算法性能。SPNE 算法采用重新计算结果集的方式，与基于影响区域更新策略相比更新代价稍高，但由于算法采用基于最短路径扩展搜索策略，实际更新性能是能够满足要求的。传统 NE 算法同样采用重新计算策略，查询更新性能稍逊于 SPNE 算法，MKNN 算法采用基于影响区域更新策略虽然能够减少搜索空间，但磁盘 I/O 代价对更新性能造成了很大影响。图 2(b)所示为固定移动对象规模为 50k 时，近邻个数 k 变化对查询更新性能的影响。SPNE 算法、传统 NE 算法和 MKNN 算法的查询更新代价随着 k 个数的增加而呈线性增加，而 SPNE 算法表现出了良好的性能。

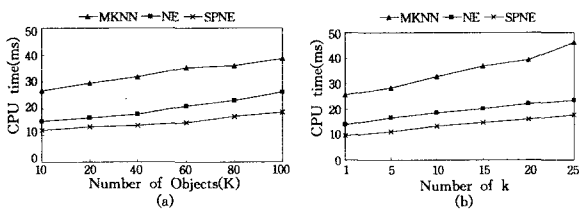

图 2 SPNE 算法更新性能比较

结束语 本文主要研究基于空间道路网络的 k 近邻查询处理技术，提出了分布式的移动对象更新策略，有效减少了服务器端的移动对象更新代价；提出了基于内存的空间道路网络邻接矩阵、最短路径矩阵和移动对象哈希表索引结构，通过在空间道路网络拓扑中集成预计算的最短路径信息，提出了基于最短路径的网络扩展搜索(SPNE)算法，并通过实验对比分析验证了算法的有效性。目前，随着多核计算技术的发展，未来工作将侧重于进行基于多核处理器的 k 近邻多用户查询并行处理框架及相关算法方面的研究。

参 考 文 献

[1] Wolfson O. Moving Objects Information Management: The Database Challenge[C]// Proceedings of the 5th International Workshop on Next Generation Information Technologies and Systems. London, 2002: 75-89

[2] Battersby S E. User-centered Design for Digital Map Navigation Tools[C]// Proceedings of the 17th International Research Symposium on Computer-based Cartography. Shepherdstown, USA, 2008

[3] Papadias D, Zhang J, Mamoulis N, et al. Query Processing in

Spatial Network Databases[C]// Proc. of 29th Intl. Conf. on Very Large Data Bases, VLDB. 2003

[4] Kolahdouzan M R, Shahabi C. Voronoi-Based k Nearest Neighbor Search for Spatial Network Databases[C]// Proc. of 30th Intl. Conf. on Very Large Data Bases, VLDB. 2004

[5] Cho H-J, Chung C-W. An Efficient and Scalable Approach to CNN Queries in a Road Network[C]// Proc. of 31th Intl. Conf. on Very Large Data Bases, VLDB. 2005

[6] Wang Haojun, Zimmermann R. Location-based Query Processing on Moving Objects in Road Networks[C]// Proc. of 29th Intl. Conf. on Very Large Data Bases, VLDB. 2007

[7] Johnson R, Hardavellas N, Pandis I, et al. To Share or Not to Share[C]// Proceedings of the International Conference on Very Large Databases, VLDB. 2007

[8] Mouratidis K, Yiu M L, Papadias D, et al. Mamoulis. Continuous nearest neighbor monitoring in road networks[C]// Proc. of 28th Intl. Conf. on Very Large Data Bases, VLDB. 2006

[9] Brinkhoff T. A Framework for Generating Network based Moving Objects[J]. GeoInformatica, 2002, 6(2): 153-180

(上接第 151 页)

前半部分使用静态调度，后半部分使用 guided 调度，这在一定程度上减少了调度开销，因此在线程数为 3,4 时，其调度性能优于 guided 调度。然而，结合 dynamic 调度以及 guided 调度优势的 trapezoid 调度，其调度块具有线性减少的特性，在尽量减少调度开销的同时，也在努力缓解负载不均衡性，因此在线程数为 3,4 时，其调度性能要优于 new_guided 调度，仅次于 dynamic 调度。

从表 3 可以看出，对于递减循环结构而言，采用 guided 和 static 调度时性能最差。依据 guided 调度的原理，在循环任务划分的过程中，开始时划分大的调度块，后期时划分小的调度块。同时，考虑到递减循环结构中循环体大小的递减特性，虽然 guided 调度在减少调度开销以及缓解负载不均衡方面做出了努力，但是在这种循环结构特性下，采用 guided 调度必然导致在开始时获得循环迭代的线程任务量很大，在后期时获得循环迭代的线程任务量很小，从而产生严重的负载不均衡性。确定 α 值为 50 的 new_guided 调度策略在一定程度上减少了调度开销，其调度性能要优于 guided 调度。然而，结合 dynamic 调度和 guided 调度优势的 trapezoid 调度在每次调度时的任务块大小具有线性递减特性，而且此调度策略实现简单，亦无需确定 α 值。因此，trapezoid 调度性能要优于 guided 调度，与 new_guided 调度性能相当。

对于广泛的应用程序而言，从图 1 可以看出，针对每种循环结构，随着线程数的增加，加速系数也在增加。然而，为了获得更好的加速系数(也可以表述为 OpenMP 多线程程序并行化性能)，不能无限制地增加线程数，需要考虑与线程相关联的开销，如线程管理开销、访存冲突问题、保存与恢复寄存器状态的开销、保存与恢复线程 cache 状态的开销等。

结束语 对于 OpenMP 多线程程序而言，需要通过良好的调度策略来将并行任务进行合理划分并分配于各线程上，从而在调度开销以及负载均衡方面获得好的权衡，最终提高程序运行性能。虽然指数调度在减少调度开销以及缓解负载不均衡方面做出了努力，但是结合动态调度以及指数调度优势的梯式调度采用线性递减的调度块方案，能够在调度开销以及负载均衡方面获得更好的权衡。本文将梯式调度策略在 OMPi 编译器进行实现，并在双核平台上，针对不同的循环结构采用不同的调度策略进行了实验评测。对于规则循环结构以及递增循环结构而言，trapezoid 与 dynamic, guided, new_guided 调度性能差别不大；对于随机循环结构而言，在线程数为 3,4 时，与 guided 调度相比，trapezoid 调度表现出更好的性能，仅次于 dynamic 调度；对于递减循环结构，trapezoid 调度性能优于 guided 调度，与 new_guided 相当。调度开销和负载均衡是相对的概念，也是影响 OpenMP 多线程程序运行性能的重要因素。为了提高程序运行性能，就需要在调度开销以及负载均衡方面做很好的权衡。

参 考 文 献

[1] OpenMP Architecture Review Board. OpenMP. Application Program Interface Version 2.5[S/OL]. http://www.openmp.org, 2005

[2] Tzen T H, Ni L M. Trapezoid self-scheduling: a practical scheduling scheme for parallel compilers[J]. IEEE Transactions on Parallel and Distributed Systems, 1993, 4(1): 87-98

[3] 周伟明. 多核编程的几个难题及其应对策略[EB/OL]. http://blog.csdn.net/drzhouweiming/archive/2007/04/10/1559698.aspx, 2007

[4] Kang Su Gatlin, Isensee P. Reap the Benefits of Multithreading without All the Work[EB/OL]. http://msdn2.microsoft.com/zh-cn/magazine//cc163717(en-us).aspx, 2008

[5] Julita C, Alejandro D, Jesús L. Dynamic Load Balancing of MPI+OpenMP Applications[C]// Proceeding of the 2004 International Conference on Parallel Processing. 2004: 195-202

[6] Chronopoulos A T, Andonie R, Benche M, et al. A Class of Loop Self-Scheduling for Heterogeneous Clusters[C]// Proceedings of the 2001 IEEE International Conference on Cluster Computing. 2001: 282-291

[7] Li H, Tandri S, Stumm M, et al. Locality and Loop Scheduling on NUMA Multiprocessors[C]// Proceedings of the 1993 International Conference on Parallel Processing. 1993: 140-147

[8] Polychronopoulos C D, Guided K D J. Self-scheduling: a practical scheduling scheme for parallel supercomputers[J]. IEEE Transactions on Computers, 1987, 36(12): 1425-1439

[9] Hummel S F, Schonberg E, Flynn L E. Factoring: a method scheme for scheduling parallel loops[J]. Communications of the ACM, 1992, 35(8): 90-101

[10] Yang Chao-Tung, Chang Shun-Chyi. A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters[J]. Journal of Information Science and Engineering, 2004, 20(2): 263-273

[11] 刘胜飞，张云泉. 一种改进的 OpenMP Guided 调度策略研究[J]. 高性能计算发展与应用，2009，26(1)：36-42

[12] Dimakopoulos V V, Alkis G. The OMPi OpenMP/C Compiler [C]// Proceeding of the 10th Panhellenic Conference on Informatics. 2005: 153-162

[13] 赖建新，胡长军，赵宇迪，等. OpenMP 任务调度开销及负载均衡分析[J]. 计算机工程，2006，32(18)：58-60