软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(5):986−995 [doi: 10.3724/SP.J.1001.2011.03811]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

无锁同步的细粒度并行介度中心算法*

涂登彪[1,2+], 谭光明[1], 孙凝晖[1]

[1](中国科学院 计算技术研究所 计算机系统结构重点实验室,北京 100190)
[2](中国科学院 研究生院,北京 100049)

Fine-Grained Parallel Betweenness Centrality Algorithm Without Lock Synchronization

TU Deng-Biao[1,2+], TAN Guang-Ming[1], SUN Ning-Hui[1]

[1](Key Laboratory of Computer System and Architecture, Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)
[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)
+ Corresponding author: E-mail: tudengbiao@ncic.ac.cn

Tu DB, Tan GM, Sun NH. Fine-Grained parallel betweenness centrality algorithm without lock synchronization. *Journal of Software*, 2011,22(5):986−995. http://www.jos.org.cn/1000-9825/3811.htm

Abstract: Through a joint study in architecture and application, it is found that lock synchronization in a fine-grained parallel betweenness centrality (BC) program poses an obstacle for the efficient execution of parallel architectures. This paper proposes a data-centric parallel algorithm that eliminates lock synchronization. This algorithm reduces execution time and improves speeds up twice as fast on both AMD 32 core SMP and Intel 8core SMP.
Key words: betweenness centrality; lock synchronization; fine-grained parallelism; data-centric; multi-core

摘 要: 通过结合体系结构和算法进行研究发现,基于锁的同步机制是细粒度并行介度中心(betweenness centrality,简称 BC)算法在现有多核平台上高效执行的主要瓶颈.提出了一种消除锁同步的数据驱动(data-centric)并行算法,在 AMD 32 核 SMP 和 Intel 8 核 SMP 两个平台上获得了 2 倍左右的加速比.
关键词: 介度中心;锁同步;细粒度并行;数据驱动;多核
中图法分类号: TP393 **文献标识码**: A

大规模网络分析在社会网络、运输网络和生物网络等重要领域中有着广泛的应用.通常,大规模网络被抽象为图,从而与图相关的数据结构和算法很自然地被用来获得网络的关键特征和提取特定的信息.对于一个给定的真实应用,通过对网络的分析和建模,可以构造出一个抽象的图.研究表明,在大多数应用领域中,这些抽象图具有 scale free 的特点,即节点的度服从幂律分布.如蛋白质相互作用网络[1]、AIDS 传播网络[2]、恐怖犯罪网络主要参与者的识别[3,4]等.在这些应用中,介度中心(betweenness centrality)[5]是分析大规模复杂网络的广泛使用的一个定量指标.通过这个指标,可以对图中节点的重要程度进行定量分析,可以衡量一个节点在网络通信中的重要性,识别出网络中的关键节点.较高的介度中心指标表明,一个节点到其他节点有相对较多的最短路径,

* 基金项目: 国家自然科学基金(60803030, 60633040, 60921002, 60925009)
收稿时间: 2008-08-01; 修改时间: 2009-08-19; 定稿时间: 2009-12-25

或者这个节点位于其他两个节点最短路径上的可能性很大.例如,通过分析蛋白质相互作用网络(protein-interaction network,简称 PIN)的结构,可以获得两个酵母蛋白对的相互作用在基因数据库的可达性[6,7].总之,介度中心的计算是网络分析的一个重要操作,它占了大部分的计算时间.所以,对这个算法进行并行加速具有重要意义.

1 相关工作

由于网络分析的重要性,目前在并行和优化方面有不少重要的研究,提出了很多基于 PRAM 模型的并行算法[8−10].然而,在当前的并行计算机上对图的算法进行并行和优化相当困难,是一个很有挑战性的问题.Bader 等人[11]分析了复杂网络中最常用的几个中心介度指标,提出了 BC 算法在高端 SMP 和多线程体系结构上的一个快速并行实现.他们的工作只是简单地开发了两级显式并行,并没有对访存和同步作特别的优化.

BC 算法的核心是图论中的典型算法——宽度优先遍历(BFS)算法,为了提高稀疏图的局部性,一般使用一种空间有效的邻接数组来存储图.Bader 等人[12]在 Cray MTA-2 多线程体系结构设计了一种多线程 BFS 算法,这种算法为了得到较好的加速比,在体系结构的特征和锁同步之间作了一些折中考虑.由于 Cray MTA-2 只有一层内存结构,他们设计的算法能够适应具有显式存储结构的众核体系结构.Park 等人[13]对 4 个基本的图算法针对 cache 作了相关的优化,对于稠密图,集中对图的邻接矩阵的布局和基于 cache 层次结构使用 cache-oblivious 技术来优化内存系统的性能.在大规模分布存储并行计算机上,消息传递的有效性对提高并行算法的扩展性有很大的影响,Yoo 等人[14]在 BlueGene/L 上提出了一个可扩展的分布式 BFS 方案.他们注意到,当对二维邻接矩阵进行合适的分块后,能够得到最优的内存使用率.但是,这项技术是为存储稠密图的邻接矩阵设计的,对于具有幂律分布的稀疏图则不适用.文献[15,16]介绍了 BFS 算法在多核平台上的最新进展,论述了 BFS 算法在 STI Cell 上的并行化和存在的挑战.在其算法中,使用 DMA 来隐藏访存延迟,但把图的节点划分成不相交的子集,需要额外的预处理.也就是说,需要把节点显式地映射到 SPU.该方法和 Yoo 等人对二维邻接矩阵进行划分的方法很类似,需要大量的集合通信操作.Tan[17]在 IBM-Cyclops64 的模拟器上作了 BC 进行了性能分析,有比较好的性能提高.

2 BC 算法简介

为了表述的完整性,下面对串行 BC 算法进行简单描述(算法的详细描述参见文献[18]).

节点数和边数分别用 n,m 表示的图 $G=(V,E)$,其中,V 和 E 分别是节点和边的集合,在有权图中,边集合 E 中的每条边 e 的权用一个正整数 $w(e)$表示,在无权图中,$w(e)=1$.定义从节点 s 到节点 t 的路径为一系列的边$\langle v_i,v_{i+1}\rangle$,$0\leqslant i\leqslant l$,$l$ 为路径经过的边数,$v_0=s$ 和 $v_l=t$.定义路径的长度为路径上所有边的权重之和,用 $d(s,t)$表示节点 s 和节点 t 之间的最短路径的长度.σ_{st} 表示节点 s 和节点 t 之间的最短路径数,$\sigma_{st}(v)$表示经过节点 v 的节点 s 和节点 t 之间的最短路径数.节点 v 的介度中心定义如下:

$$BC(v)=\sum_{s\neq v\neq t\in V}\frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1}$$

对于 n 个节点、m 条边的图 G,用公式(1)直接计算介度中心的时间复杂度为 $O(n^3)$.Brandes 等人[18]提出了一种改进算法,使得计算介度中心的时间复杂度为 $O(nm)$.Brandes 算法的主要贡献在于消除了求节点对相关性的冗余求和操作.

Brandes 算法描述如下:对于一个给定的源顶点,找出这个源节点到其他节点的所有最短路径,用来计算节点对之间的相关性.设从源节点 s 开始遍历时,节点 v 的前驱集合为

$$P_s(v)=\{u\in V:\{u,v\}\in E,d_G(s,v)=d_G(s,u)+w(u,v)\} \tag{2}$$

为了消除所有节点对相关性的冗余求和,定义节点 s 对节点 v 的相关性为

$$\delta_s(v)=\sum_{t\in V}\frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3}$$

显然,$\delta_s(v)$是 $BC(v)$的部分和.节点 $v\in V$ 的介度中心可以表示为

$$BC(v)=\sum_{s\neq v\in V}\delta_s(v) \tag{4}$$

Brandes 算法的关键之处在于观察到部分和服从递归关系,从而消除了冗余的计算:

$$\delta_s(v)=\sum_{w:v\in P_s(w)}\frac{\sigma_{sv}}{\sigma_{sw}}(1+\delta_s(w)) \tag{5}$$

在大规模网络分析中,为了减少内存空间的使用,一般用带索引的邻接数组表示图 G 的数据结构.图 1 是用一个带索引的邻接数组表示图的一个简单的例子,它由一个索引数组和一个邻接数组组成.前驱集合 P 保存了 BFS 遍历的轨迹,它保存在另一个邻接数组中.参数 d,σ,δ和 bc 分别用数组来实现,然而对 d,σ,δ等 3 个数组的引用非常依赖于数组 G,P.算法有两个主要阶段:BFS 遍历和回溯求和.在 *BFS* 阶段,从 n 个源节点 $s\in V$ 开始做 BFS/Dijkstra 遍历.在遍历的过程中,记录节点 $v\in V$ 的前驱集合 $P_s(v)$和通过前驱 w 的最短路径数$\delta_s(w)$.在回溯求和阶段,对于每一个从源节点 $s\in V$ 开始的遍历,根据记录在 $P_s(v),\delta_s(w)$中的信息利用递归关系式(5)求得所有节点 $v\in V$ 的相关性.

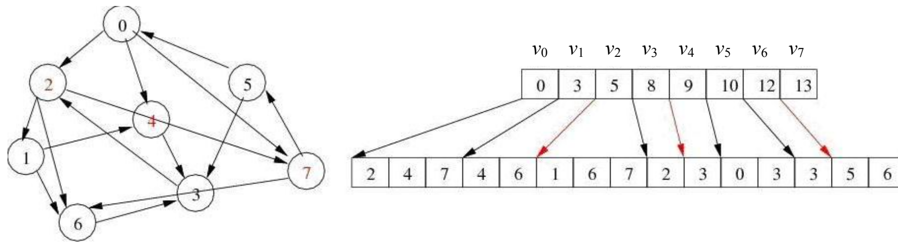


Fig. 1 Adjacent array of graph

图 1 图的邻接数组

3 并行 BC 算法

在这一节中,介绍了我们对 BC 算法并行化的研究.首先分析了 BC 算法本身的并行性,并分析了锁同步对性能的影响,然后,我们提出了一个消除锁同步的并行策略.

3.1 并行问题

BC 算法从每个源节点开始做宽度优先遍历,然后回溯累加得到介度中心.这个过程可以开发 3 种粒度的并行性:

(1) 粗粒度并行.可以把每个从源节点开始的遍历及回溯累加过程看作一个任务,算法需要 n 个任务,求得每个节点的部分中心介度值.最后,把每个任务的部分 *BC* 值累加,得到每个顶点的最终 *BC* 值.显然,这 n 个任务可以并行处理,需要一个并行规约操作得到最终的值.该并行策略需要复制 n 份图的数据结构以及 n 份 P,d,σ,δ数组.对于一个大规模图,这些数据结构需要的内存空间很容易超过物理内存的大小,尤其是多核和众核体系结构的共享内存容量.

(2) 中粒度并行.宽度优先搜索按层生成遍历树,在遍历当前层所有顶点的邻接点时,把第 1 次访问到的邻接点作为下一层的节点.可以把当前层每一个节点遍历它的邻接点的过程看作一个任务,如果当前层的任何两个节点没有共同的邻接点,所有的任务就可以完全并行处理;否则,需要同步机制来强制顺序性.此外,每一个任务的执行时间依赖于每个节点的邻接点的个数,而稀疏的 scale free 图中,每个节点的度的分布是不均衡的,负载平衡在一级并行中很难获得.

(3) 细粒度并行.当遍历某个节点的邻接点时,对每个邻接点的操作可以并行化.显然,可获得的并行度依赖于每个节点的度.在稀疏 scale free 图中,只有极少数的节点有较高的度,因此,在这一级能够开发的并行性有限.

在回溯求和阶段,求$\delta(v)$需要锁同步机制.算法 1 用伪码描述了 HPCS Benchmark—SSCA2 中并行 BC 算法的实现.把访问队列 Q 中顶点 v 的邻接点的过程定义为节点 v 的扩展操作,即算法 1 中第 4 行~第 12 行的操作.在细粒度同步算法中,当两个节点有共同的邻接点时,需要进行冲突处理(如图 1 中节点 2 和节点 7 对 6 号节点的访问).采用锁处理冲突时,通常锁的个数和节点的个数一样多,锁同步带来很大的内存开销,导致本地局部内存或 cache 不能把它保存下来.更糟糕的是,保存锁的数组的访存模式依赖于节点的访存模式,因此,对锁的访问也是动态非连续的.

算法 1. 并行 BC 算法.

```
1.  while Q is not empty {
2.  #pragma omp for shedule(dynamic)
3.  for each v in Q {
4.  for each neighbors w of v {
5.  omp_set_lock(& lock[w]);
6.  if (d[w]<0)
7.  {   enqueue w→Q; d[w]=d[v]+1;}
8.  if (d[w]==d[v]+1) {
9.  sigma[w]+=sigma[v];
10. append v→P[w];
11. }
12. omp_unset_lock(& lock[w]);
13. } } }
14. #pragma omp for schedule(dynamic)
15. for each w in Q {   //Q is emulated as stack
16. for each predecessor v of w {
17. omp_set_lock(& lock[v]);
18. delta[v]+=(sigma[v]/sigma[w])(1+delta[w]);
19. omp_unset_lock(& lock[w]);
20. }
21. if (w!=s)
22. BC[w]+=delta[w];
23. }
```

3.2 消除锁同步

根据并行策略的不同,我们把并行算法分为 task-centric 和 data-centric 两种类型,算法 1 中的各线程对队列 Q 中节点 v 执行扩展操作就是一种 task-centric 类型的操作.本文采用 data-centric 策略保证并行任务由数据分布来决定.在我们设计的并行 BC 算法的 BFS 遍历和回溯求和两个阶段,分别把 data-centric 细分为 vertex-centric 和 edge-centric 两种类型.下面详细介绍运用 data-centric 策略设计细粒度并行算法,消除锁同步.

3.2.1 数据结构

首先对队列 Q 和前驱集合 P 的数据结构进行了重新设计.

- 队列 Q

假定有 p 个线程,把节点集合 V 分成 p 个不相交的子集 V_i,$0\leqslant i<p$,把队列 Q 分成线程共享的 p 个子队列 Q_i,$0\leqslant i<p$.线程 i 对属于集合 V_i 中的节点进行处理,往子队列 Q_i 中添加处理好的节点.线程 i、集合 V_i、子队列 Q_i 是一一对应的.也就是说,子队列 Q_i 中的节点都属于集合 V_i,子队列 Q_i 中的节点都是由线程 i 添加的.

- 前驱集合 P

为了保存回溯路径,需要记录每个节点的前驱集合.算法 1 为每一个节点动态维护一个链表,用来记录前驱集合.因为多个节点可能拥有同一个父节点,当多个线程把父节点插入到前驱集合时,需要一个同步操作来避免写冲突.此外,动态内存管理也会带来额外的开销.本文提出了一种记录前驱节点的新方法.如图 2 所示,我们采用一个三元数组 P 来记录前驱集合,数组的每一个元素为一个三元组〈父节点,子节点,层〉,假定边$\langle v,w\rangle$的节点 v 和 w 分别在第 $i-1$ 层和第 i 层,则在数组中记入一个元素$\langle v,w,i\rangle$.

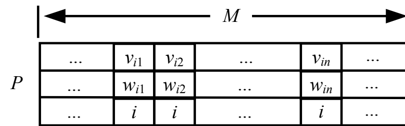


Fig.2 Triple array shared by all threads

图 2 被所有线程共享的一个三元组数组

3.2.2 算法流程

BFS 遍历阶段,源节点 s 所在的层为第 0 层.在初始化后,源节点 s 加入到对应的子队列中.然后,线程 i 扫描各子队列,得到当前层的所有节点 v;对节点 v 执行扩展操作,遍历节点 v 的所有邻接点 w;同时,对遍历经过的边依次编号.假设节点 v 经过编号为 E_i 的边到达邻接点 w,当节点 w 属于集合 V_i 时,则由线程 i 处理节点 w.如果源节点 s 到节点 w 的距离为 $d(s,w)=d(s,v)+w(v,w)$,则边$\langle v,w\rangle$在源节点 s 到节点 w 的最短路径上.为了便于回溯,把$\langle v,w,k\rangle$记入三元数组中下标为 E_i 的位置,k 为节点 w 所在的层.如果节点 w 是第 1 次被访问,则把节点 w 加到子队列 Q_i 的下一层.*BFS* 遍历阶段的并行执行是由子队列 Q_i 中的节点驱动的,所以是 vertex-centric 类型.回溯求和阶段,从最后一层开始按层扫描三元数组,根据公式(5),由父节点所在集合对应的线程更新δ值,直到扫描完所有的边为止.回溯求和阶段的并行执行是由三元数组中的边驱动的,所以是 edge-centric 类型.

新算法的伪码描述见算法 2.

算法 2. 无锁同步的并行 BC 算法.

```
1.   #pragma omp parallel private(Tr,L)
2.   {
3.     tid=omp_get_thread_number();
4.     while each Q[i] is not empty {
5.       for each v in Q[i] {   //0≤i<p
6.         for each neighbors w of v {
7.           Tr++;
8.         if w in V[tid] {
9.             if (d[w]<0)
10.            {enqueue w→Q[tid]; d[w]=d[v]+1;}
11.            if (d[w]==d[v]+1) {
12.              sigma[w]+=sigma[v]; P[Tr].parent=v;
13.              P[Tr].child=w;        P[Tr].level=L;
14.      } } } }
15.      L++;
16.      #pragram omp barrier
17.    }
18.    while L≥0 {   //backtrace level by level
19.      for each edge(v,w) in level L {
20.        if v in V[tid]
```

```
21.             delta[v]+=(sigma[v]/sigma[w])(1+delta[w]);
22.          }
23.          L--;
24.          #pragram omp barrier
25.      }
26.      #pragma omp for
27.      for each v {
28.         BC[v]+=delta[v];
29.   } }
```

3.2.3 算法分析

显然,算法 1 和算法 2 的时间复杂度都为 $O(nm)$.下面从内存开销和冲突处理两个方面对算法进行分析.

在 BFS 遍历阶段,可以证明有下面的定理成立:

定理 1. 在 vertex-centric 算法中,给定节点的一个划分 $V_1,V_2,\ldots,V_p$,子队列 Q_i 的长度小于或等于对应集合 V_i 中元素的个数,$0\le i<p$.这里,$|V_1|+|V_2|+\ldots+|V_p|=|V|$,符号$|\cdot|$表示集合中元素的个数.

证明:由于子队列 Q_i 和子集合 V_i 一一对应,显然,子队列 Q_i 中的节点都是子集合 V_i 中的元素.而且子队列 Q_i 中的节点只在第 1 次被访问时才被加入队列中,所以子队列 Q_i 中的节点是唯一的,即同一个节点不会被反复加入到子队列 Q_i 中.所以,子队列 Q_i 的长度小于或等于对应集合 V_i 中元素的个数. □

由定理 1 可知,子队列 Q_i 的内存开销和算法 1 中队列 Q 一样.但是,子队列 Q_i 是按层存放的,所以需要 p 个长度为最大层数的数组对子队列 Q_i 分层.根据算法 1 中的数据,最大的层数不会超过 $\sqrt{n}$,所以额外的开销为 $O\left(p\sqrt{n}\right)$.

在回溯累加阶段,为了保存前驱节点,我们设计的三元数组 P 需要的内存空间为 $O(3m)$.算法 1 中为了保存节点的前驱节点,需要的内存开销为 $O(3n+m)$.如果 $m=8n$,则算法 1 需要的内存开销是 $O(11n)$,算法 2 需要的内存开销是 $O(24n)$.如果考虑到算法 1 在并行时锁同步带来的大量内存开销,算法 2 相对算法 1 在内存开销方面并没有明显的增加.

在 BFS 遍历阶段和回溯求和阶段,图的数据结构是只读的,并行执行时,不会因为数据冲突破坏数据的一致性.d,σ,δ,P,Q 这几个重要的数据结构可能会有多个线程同时进行读写,所以需要注意避免冲突,保持数据的一致性.在 BFS 遍历阶段,由于是按层遍历的,当遍历到第 i 层时,d,σ,P,Q 这几个数组中的第 0 层~第 $i-1$ 层的数据都已经写入.第 i 层数据的更新只依赖第 $i-1$ 层的数据,避免了先写后读冲突或先读后写冲突的产生,而执行写入操作的线程由节点所属集合唯一确定,从而避免了写写冲突.在回溯累加阶段,对δ的更新也是分层的,可以采用和 BFS 遍历阶段相同的方法来避免冲突.

3.2.4 实例分析

本节以一个无向图为例对本文提出的算法做简单的演示,图 3 为从一个源节点出发做宽度优先搜索(BFS)时得到的第 $i-1$ 层节点和第 i 层节点以及从第 $i-1$ 层~第 i 层的边.

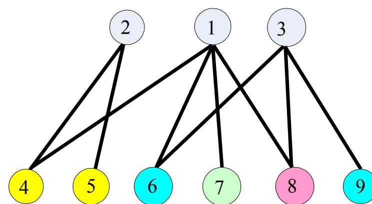


Fig.3 Edges from level $i-1$ to level i

图 3 第 $i-1$ 层到第 i 层的边

假设有 4 个线程并行执行,第 i 层的节点分别属于 4 个不同的集合:节点 4 和节点 5 属于集合 V_1,节点 6 和节点 9 属于集合 V_2,节点 7 属于集合 V_3,节点 8 属于集合 V_4.线程 i(i=1,2,3,4)只修改集合 V_i 中节点的相关的数据.在 BFS 阶段,4 个线程并行当遍历完前 i–1 层节点后,此时数据结构的内容如图 4 所示.

从第 i–1 层开始遍历第 i 层时,4 个线程并行读数组 Q 中第 i–1 层的节点:节点 1、节点 2、节点 3,然后从每个节点开始搜索第 i 层的节点,并修改第 i 层中节点的对应数据,如数组 *dist* 和数组 *sigma* 中对应的数据.如,线程 1 依次从节点 1、节点 2、节点 3 开始搜索下一层的节点,因只有节点 4 和节点 5 属于和线程 1 对应的集合 V_1,故只有线程 1 才能修改和节点 4、节点 5 相关的数据.虽然其他线程也会遍历到节点 4 和节点 5,但因为节点 4 和节点 5 不属于其他线程对应的集合,所以其他线程不能修改和节点 4 和节点 5 相关的数据.通过这种方式,避免了对节点 4 和节点 5 的写冲突.又由于修改第 i 层数据只依赖从第 i–1 层读到的数据,所以不会出现先读后写冲突和先写后读冲突,从而避免了对节点加锁的操作.遍历完第 i 层后,数据结构对应的内容如图 5 所示.

		Level i–1			Level i						
Q	...	1	2	3							...
dist	...	2	2	2							...
sigma	...	4	3	5							...
delta	...										...
	...										...
P	...										...
	...										...

Fig.4 Data structure after visit level i in BFS phase

图 4 BFS 阶段遍历第 i 层后的数据结构

		Level i–1			Level i						
Q	...	1	2	3	4	5	6	7	8	9	...
dist	...	3	3	3	4	4	4	4	4	4	...
sigma	...	4	3	5	7	3	9	4	9	5	...
delta	...										...
	...	1	1	1	1	2	2	3	3	3	...
P	...	4	6	7	8	4	5	6	8	9	...
	...	*i*	*i*	*i*	*i*	*i*	*i*	*i*	*i*	*i*	...

Fig.5 Data structure before visit level i in BFS phase

图 5 BFS 阶段开始遍历第 i 层前的数据结构

在回溯阶段,节点从第 i 层回溯到第 i–1 层前对应的数据结构如图 6 所示.用和 BFS 阶段相同的方法,可以避免冲突的产生,这里不再赘述.回溯完第 i 层后的数据结构如图 7 所示.

		Level i–1			Level i						
Q	...	1	2	3	4	5	6	7	8	9	...
dist	...	3	3	3	4	4	4	4	4	4	...
sigma	...	4	3	5	7	3	9	4	9	5	...
delta	...				1.2	1.4	3.2	1.8	2.6	2.2	...
	...	1	1	1	1	2	2	3	3	3	...
P	...	4	6	7	8	4	5	6	8	9	...
	...	*i*	*i*	*i*	*i*	*i*	*i*	*i*	*i*	*i*	...

Fig.6 Data structure before backtrace level i–1 in backtrace phase

图 6 回溯阶段回溯第 i–1 层前的数据结构

		Level i−1			Level i						
Q	...	1	2	3	4	5	6	7	8	9	...
dist	...	3	3	3	4	4	4	4	4	4	...
sigma	...	4	3	5	7	3	9	4	9	5	...
delta	...	5.6	4.4	7.5	1.2	1.4	3.2	1.8	2.6	2.2	...
	...	1	1	1	1	2	2	3	3	3	...
P	...	4	6	7	8	4	5	6	8	9	...
	...	i	i	i	i	i	i	i	i	i	...

Fig.7 Data structure after backtrace level i−1

图 7 回溯阶段回溯第 i−1 层后的数据结构

4 性能评价

本节介绍实验方法和实验结果.通过与基于OpenMP锁同步机制实现的BC算法比较,本文提出的消除锁同步的细粒度并行算法有近 2 倍的加速比.此外,还分析了影响可扩展性的因素,作为下一步研究的参考.

4.1 实验方法

实验平台是两个主流的多核处理器组成的 SMP 系统,分别是 Intel 至强系列 E5530 服务器和 AMD 的 32 核胖节点.两个平台的配置见表 1.

Table 1 Configuration of experimental platform

表 1 实验平台的配置

	Intel E5530	AMD smp
CPU model name	Intel(R) Xeon(R) CPU E5530	Quad-Core AMD Opteron(tm) processor 8378
Frequency	2.4GHz	2.4GHz
Number of cores	8 (2P4CoreX2Thread)	32 (8p4c)
Memory	16GB	128GB
Kernel version	Linux 2.6.9-67.Elsmp	Linux 2.6.9-67.Elsmp
Compiler	PGI 9.0	PGI 9.0
Flags	−O3 −mp	−O3 −mp

HPCS Benchmark 集中的 SSCA2 用 OpenMP 实现了 BC 算法,是目前文献中提到的性能最好的并行算法之一.本文提出的新算法以它作为性能比较的参考.

4.2 实验结果

表 2 和表 3 列出了 BC 算法在 Intel 8 核平台和 AMD 32 核平台上在不同规模下的执行时间及新算法相对于原算法的加速比.表中 Orig 表示 HPCS SSCA2 中基于锁同步实现的并行 BC 算法,Opt 表示本文提出的无锁同步的并行 BC 算法,图的规模 S 表示图的节点数为 2^S,边的条数为 2^{3+S}.在表的最后一列,我们给出了平均加速比.从表中的数据可以看出,与 HPCS SSCA2 中 BC 算法相比,本文提出的无锁同步的并行 BC 算法在这两个实验平台上有 2 倍左右的加速比.

在其他多核平台,如 8 核 AMD Barcelona 和 Intel Clovertown,16 核的曙光 5000A 节点,本文提出的新算法相对于原算法也有 2 倍左右的加速比,限于篇幅,这里不再给出相应的实验结果.

文中提出的并行算法因为消除了锁同步,相对于原并行算法有很好的加速比.然而随着核的增加,两个并行算法都不能取得较好的可扩展性.导致算法可扩展性不好的主要因素在于,BC 算法的计算访存比很低,可以开发出的计算并行度不高.另一方面,存储墙(memory wall)的存在导致这类局部性不好的负载,很难找到满足算法

需要的存储系统设计.随着问题规模的增加,本文提出的算法相对原算法的加速比逐渐减小.这主要是因为随着问题规模的增加,原算法发生锁冲突的概率随之减小,锁同步开销对程序性能的影响也随之减小.但从实验数据来看,即使问题规模增大了,本文提出的算法相对于原算法依然有加速比.

Table 2 Experimental results on 8-core Intel platform

表 2 Intel 8 核平台上的实验结果

	2 threads			4 threads			8 threads		
S	Orig	Opt	**Sp**	Orig	Opt	**Sp**	Orig	Opt	**Sp**
12	2.07	1.07	**1.9**	1.46	0.94	**1.6**	1.64	0.9	**1.8**
13	4.10	2.05	**2.0**	2.79	1.87	**1.5**	3.01	1.69	**1.8**
14	8.22	4.13	**2.0**	5.40	3.67	**1.5**	5.77	3.23	**1.8**
15	17	8	**2.1**	11	7	**1.6**	11	6	**1.8**
16	38	18	**2.1**	23	15	**1.5**	22	13	**1.7**
17	82	39	**2.1**	48	33	**1.5**	45	28	**1.6**
18	183	83	**2.2**	108	69	**1.6**	94	56	**1.7**
19	414	192	**2.1**	231	138	**1.7**	197	117	**1.7**
20	899	444	**2.0**	511	333	**1.5**	406	260	**1.6**
21	1 945	1 020	**1.9**	1 096	772	**1.4**	875	623	**1.4**
22	4 252	2 230	**1.9**	2 373	1 621	**1.5**	1 824	1 330	**1.4**
Avg.			**2.0**			**1.5**			**1.7**

Table 3 Experimental results on 32-core AMD fat node

表 3 AMD 32 核胖节点上的实验结果

	2 threads			4 threads			8 threads			16 threads			32 threads		
S	Orig	Opt	**Sp**	Orig	Opt	**Sp**	Orig	Opt	**Sp**	Orig	Opt	**Sp**	Orig	Opt	**Sp**
12	4.32	1.78	**2.4**	4.18	1.66	**2.5**	3.54	2.03	**1.7**	4.83	2.36	**2.0**	7.10	3.00	**2.4**
13	9.57	4.11	**2.3**	8.03	3.10	**2.6**	6.78	3.6	**1.9**	8.59	4.1	**2.1**	11.8	6.12	**1.9**
14	16.5	6.27	**2.6**	15.3	5.52	**2.9**	12.7	6.55	**1.9**	16.5	8.03	**2.1**	23.1	9.7	**2.4**
15	42	18	**2.3**	31	13	**2.4**	24	11	**2.2**	30	14	**2.2**	43	25	**1.7**
16	76	38	**2.0**	65	28	**2.3**	50	28	**1.8**	59	34	**1.7**	72	46	**1.6**
17	172	89	**1.9**	135	67	**2.0**	102	70	**1.5**	113	80	**1.4**	163	132	**1.2**
18	405	210	**1.9**	295	169	**1.7**	209	154	**1.4**	206	164	**1.3**	308	293	**1.1**
19	998	528	**1.9**	626	369	**1.7**	430	324	**1.3**	399	349	**1.2**	601	573	**1.1**
20	2 391	1 321	**1.8**	1 392	904	**1.5**	868	698	**1.2**	882	721	**1.2**	1184	1 189	**1.1**
21	5 317	3 098	**1.7**	2 951	2 092	**1.4**	1 793	1 566	**1.1**	1 700	1 533	**1.1**	2 308	2 168	**1.1**
22	11 584	6 708	**1.7**	6 219	4 366	**1.4**	3 753	3 497	**1.1**	3 550	3 397	**1.1**	4 706	3 962	**1.1**
Avg.			**2.0**			**2.0**			**1.6**			**1.6**			**1.5**

5 结 论

介度中心算法是大规模网络分析的重要算法.本文提出了一种以数据驱动消除锁同步的细粒度并行介度中心算法,并在多核处理器上实现了该算法.实验结果表明,在主流的多核处理器构建的 SMP 平台上,本文提出的算法相对于基于锁同步的算法有 2 倍左右的性能提高.从实验分析可以看出,BC 算法的特征和传统的科学计算不同,在多核/众核平台上求解非规则类应用的效率比较低,需要深入分析算法和数据结构,更好地利用计算机体系结构的特征以提高性能.

References:

[1] del Sol A, Fujihashi H, O'Meara P. Topology of small-world networks of protein—Protein complex structures. Bioinformatics, 2005,21(8):1311−1315. [doi: 10.1093/bioinformatics/bti167]

[2] Liljeros F, Edling CR, Amaral LAN, Stanley HE, Aberg Y. The Web of human sexual contacts. Nature, 2001,411(6840):907−908. [doi: 10.1038/35082140]

[3] Coffman T, Greenblatt S, Marcus S. Graph-Based technologies for intelligence analysis. Communications of the ACM, 2004,47(3): 45−47. [doi: 10.1145/971617.971643]

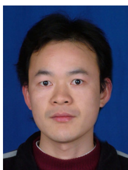
[4] Krebs VE. Mapping networks of terrorist cells. Connections, 2002,24(3):43−52.

[5] Freeman LC. A set of measures of centrality based on betweenness. Sociomtry, 1977,40(1):35−41. [doi: 10.2307/3033543]

[6] Joy M, Brock A, Ingber D, Huang S. High-Betweenness proteins in the yeast protein interaction network. Journal of Biomed Biotechnol, 2005,2005(2):96−103. [doi: 10.1155/JBB.2005.96]
[7] Newman MEJ, Girvan M. Finding and evaluating community structure in networks. Physical Review E, 2004,69(2): 2611301−2611315. [doi: 10.1103/PhysRevE.69.026113]
[8] Crauser A, Mehlhorn K, Meyer U, Sanders P. A parallelization of dijkstras shortest path algorithm. Lecture Notes in Computer Science, 1998,1450:722−731. [doi: 10.1007/BFb0055823]
[9] Grama AY, Kumar V. A survey of parallel search algorithms for discrete optimization problems. ORSA Journal on Computing, 1993,7. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.9937
[10] Klein PN, Subramanian S. A randomized parallel algorithm for single-source shortest paths. Journal of Algorithms, 1997,25(2): 205−220. [doi: 10.1006/jagm.1997.0888]
[11] Bader DA, Madduri K. Parallel algorithms for evaluating centrality indices in real-world networks. In: Feng WC, ed. Proc. of the 35th Int'l Conf. on Parallel Processing (ICPP 2006). Washington: IEEE Computer Society, 2006. 539−550. [doi: 10.1109/ICPP. 2006.57]
[12] Bader DA, Madduri K. Designing multithreaded algorithms for breadth-first search and *st*-connectivity on the cray mta-2. In: Feng WC, ed. Proc. of the 35th Int'l Conf. on Parallel Processing (ICPP 2006). Washington: IEEE Computer Society, 2006. 523−530. [doi: 10.1109/ICPP.2006.34]
[13] Park JS, Penner M, Prasanna VK. Optimizing graph algorithms for improved cache performance. IEEE Trans. on Parallel and Distributed Systems, 2004,15(9):769−782. [doi: 10.1109/TPDS.2004.44]
[14] Yoo A, Chow E, Henderson K, McLendon W, Hendrickson B, Aatalyurek A. A scalable distributed parallel breadth-first search algorithm on BlueGene/L. In: Proc. of the ACM/IEEE Supercomputing 2005. Washington: IEEE Computer Society, 2005. 25−44. [doi: 10.1109/SC.2005.4]
[15] Villa1 O, Scarpazza1 DP, Petrini1 F, Peinador JF. Challenges in mapping graph exploration algorithms on advanced multicore processors. In: Proc. of the 21st IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS). 2007. 1−10. [doi: 10.1109/IPDPS. 2007.370253]
[16] Gschwind M, Hofstee P, Flachs B, Hopkins M, Watanabe Y, Yamazaki T. Synergistic processing in Cell's multicore architecture. IEEE Micro, 2006,26(2):10−24. [doi: 10.1109/MM.2006.41]
[17] Tan GM. Locality and parallelism of algorithm in irregular computation [Ph.D. Thesis]. Beijing: Institute of Computing Technology, the Chinese Academy of Sciences, 2008 (in Chinese with English abstract).
[18] Brandes U. A faster algorithm for betweenness centrality. Journal of Mathematical Socialogy, 2001,25(2):163−177. [doi: 10.1080/ 0022250X.2001.9990249]

附中文参考文献:

[17] 谭光明.非规则计算中的局部性和并行性[博士学位论文].北京:中国科学院计算技术研究所,2008.



涂登彪(1979－),男,湖南湘阴人,博士,CCF学生会员,主要研究领域为高性能计算,并行算法及其优化,信息安全.

孙凝晖(1968－),男,博士,研究员,博士生导师,CCF高级会员,主要研究领域为并行体系结构,分布式操作系统,高性能计算.

谭光明(1980－),男,博士,助理研究员,CCF会员,主要研究领域为高性能算法,体系结构.