Journal of Computer Applications
计算机应用,2012,32(9):2458 -2462
ISSN 1001-9081
CODEN JYIIDU
2012-09-01
http://www.joca.cn

文章编号:1001 -9081(2012)09 -2458 -05
doi:10.3724/SP.J.1087.2012.02458

快速复杂网络聚类图形处理器并行算法

王海峰[1,2*]

(1.临沂大学 信息学院,山东 临沂 276002; 2.上海理工大学 管理学院,上海 200093)

(*通信作者电子邮箱 gadfly7@126.com)

摘 要:研究复杂网络拓扑属性的聚类算法需要处理大量节点和连接边,因此对计算性能要求高,否则无法处理现实中的表示为复杂网络的系统。利用图形处理器(GPU)的并行聚类算法是解决该问题的重要方法。利用原语技术设计并行快速聚类算法,原语法不仅降低并行算法的复杂性而且提高聚类的普适性;再从线程调度策略和缓存管理两个方面提出优化的方法来解决负载均衡和数据重用性问题。通过实验对比并行快速聚类算法与优化算法的性能,结果显示并行快速聚类优化后的算法取得较好加速比。

关键词:簇结构;复杂网络;聚类发现;图形处理器;并行算法

中图分类号:TP301.6 **文献标志码**:A

Parallel algorithms for complex network clustering with GPUs

WANG Hai-feng[1,2*]

(1. *School of Information, Linyi University, Linyi Shandong* 276002, *China*;
2. *School of Management, University of Shanghai for Science and Technology, Shanghai* 200093, *China*)

Abstract: The complex network clustering algorithm to research the topology properties of complex network needs to deal with large-scale nodes and links. Therefore, it requires higher computation performance to the large-scale complex networks that represent the complex system in reality. Hence, a parallel complex network clustering algorithm on Graphic Processing Units (GPU) based on fast Newman clustering algorithm was designed through the primitive technology that not only reduced the complexity of the parallel algorithm design but also improved the universality in various applications. Then from the thread scheduling strategies and the cache management perspectives, the optimal parallel complex network clustering algorithms were presented to deal with the load balance and the data reuse problem in computing process. The experiments results of the parallel complex network clustering algorithm and the optimal algorithms show that the optimal algorithms have better performance than the former.

Key words: cluster structure; complex network; clustering discovery; Graphic Processing Unit (GPU); parallel algorithm

0 引言

复杂网络是描述复杂系统的一种重要手段,簇结构是最重要的拓扑属性之一[1]。发现复杂网络簇结构的聚类算法对分析网络拓扑特性、理解其行为和功能、探索其隐含模式、预测未来行为具有重要意义[2]。复杂网络聚类算法主要分为两大类:基于优化的方法,如谱方法和局部搜索法;启发式方法,如最经典的 Girvan-Newman 算法[2-3]、遗传算法[4]、粒子群算法[5-6]、模块密度法[7]等。然而现实中的复杂系统规模庞大,构成的复杂网络具有大量节点和连接边。例如社会网络中的人际关系网、交通网络、Web 中的博客网等都具有海量节点和边。串行算法难以满足实际的计算性能,因此复杂网络并行聚类算法的研究逐渐被关注。图形处理器(Graphic Processing Unit, GPU)通用计算技术是一种性价比高的并行处理方案,在科学计算中大量经典算法被移植到 GPU 中以获得更高计算性能。在此以快速 Newman(Fast Newman, FN)算法[8]为研究对象,研究 CUDA 平台上的并行聚类算法,在两种不同线程调度策略下分别提出两种并行算法,然后结合线程调度策略及 GPU 缓存特性设计出两种优化算法,经修改可扩展到其他串行聚类算法中,具有一定普适性。

1 快速复杂网络聚类算法

1.1 网络模块性

FN 算法为基于局部搜索的快速复杂网络聚类算法,它以网络模块性为优化依据。Kernighan-Lin 算法、Guimera-Amaral 算法及基于模拟退火算法都属于基于模块系数的局部搜索算法。局部搜索策略的基础是网络模块性,采用 Q 函数描述复杂网络模块性,定义为网络簇内实际连接数目与随机连接情况下的期望连接数目之差。

$$Q = \sum_{i=1}^{n}\left[\frac{m_i}{m} - \left(\frac{d_i}{2m}\right)^2\right] \tag{1}$$

其中:Q 定量地描述复杂网络中簇结构的优劣,n 表示网络中簇的个数,m 表示网络连接总数,m_i 表示网络簇 i 中连接总数,d_i 表示网络簇 i 中节点度总和。

$$\Delta Q_{i,j} = \sum_{j=1}^{n-1}\left[\frac{m_j}{m} - \left(\frac{d_j}{2m}\right)^2\right] - \sum_{i=1}^{n}\left[\frac{m_i}{m} - \left(\frac{d_i}{2m}\right)^2\right] \tag{2}$$

假定网络簇 i 与 j 合并为一个新网络簇,新网络簇中簇内连接增加值表示为 m_{ij},则得到式(3):

$$\Delta Q_{i,j} = \frac{m_{ij}}{m} - \frac{d_i \times d_j}{2m^2} \tag{3}$$

1.2 Q 值局部搜索算法

局部搜索策略以网络模块性为搜索依据,从网络簇仅包含一个节点的初始解开始,搜索具有最大 $\Delta Q_{i,j}$ 值的两个网络簇并执行合并操作,直到满足收敛条件停止搜索。FN 算法采用自底向上、广度优先的层次聚类方法,输出一棵描述网络簇

收稿日期:2012-03-26;**修回日期**:2012-05-25。 **基金项目**:上海市重点学科建设项目(S30501)。

作者简介:王海峰(1976 -),男,山东临沂人,副教授,博士,主要研究方向:复杂网络、并行计算。

层次关系的树,时间复杂度为 $O(mn)$,其中 m 和 n 分别表示复杂网络的连接边数和节点数。在搜索策略中由于计算 $\Delta Q_{i,j}$ 最耗时间,整个复杂网络的 $\Delta Q_{i,j}$ 构成矩阵 $\Delta \boldsymbol{Q}$,通过 $\Delta \boldsymbol{Q}$ 矩阵和最大堆 H 来优化计算性能。将 $\Delta \boldsymbol{Q}$ 矩阵的一行元素以平衡二叉树方式存储,检索与插入元素操作的时间复杂度降为 $O(\log n)$,同时将求最大值操作的时间复杂度控制在常数范围内。假设有 n 个节点的复杂网络,下面是对 FN 串行算法的具体描述[8]:

1) 建立 n 个包含一个节点的网络簇;

2) 计算 $\Delta Q_{i,j}$,并建立 $\Delta \boldsymbol{Q}$ 矩阵;

3) 对 $\Delta \boldsymbol{Q}$ 矩阵每一行元素进行堆排序操作,并选出最大 $\Delta Q_{i,j}$;

4) 合并网络簇 i, j,更新 $\Delta \boldsymbol{Q}$ 矩阵;

5) 重复执行 2),直至复杂网络子簇数达到一定预设值。

2　并行快速聚类算法

2.1　数据结构

复杂网络存在多种逻辑存储结构,比如数组、链表、图等。

1	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	−1	−1
1	C_{21}	C_{22}	C_{23}	−1	−1	−1	−1
1	C_{31}	C_{32}	C_{33}	C_{34}	−1	−1	−1
1	C_{41}	C_{42}	C_{43}	−1	−1	−1	−1
1	C_{51}	C_{52}	C_{53}	C_{54}	C_{55}	−1	−1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	C_{n1}	C_{n2}	C_{n3}	C_{n4}	−1	−1	−1

(a) 网络簇2,4合并前

1	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	−1	−1
1	C_{21}	C_{22}	C_{23}	C_{41}	C_{42}	C_{43}	−1
1	C_{31}	C_{32}	C_{33}	C_{34}	−1	−1	−1
−1	C_{41}	C_{42}	C_{43}	−1	−1	−1	−1
1	C_{51}	C_{52}	C_{53}	C_{54}	C_{55}	−1	−1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	C_{n1}	C_{n2}	C_{n3}	C_{n4}	−1	−1	−1

(b) 网络簇4合并到2

图 1　网络簇合并示意图

2.2　子聚类合并

若当前 $\Delta Q_{2,4}$ 为最大值,需要将子聚类 C_2、C_4 合并成一个新聚类。图 1 为合并 C_2、C_4 的示意图,图 1(a) 为合并前的情况,C_2 为有效状态并包含 3 个节点元素,C_4 也是有效状态并包含3个节点元素;图1(b) 为合并后的情况,C_4 为无效状态,而 C_2 状态有效且添加了 C_4 中的3个节点变成 6 个节点的子聚类。

2.3　并行原语

在 CUDA 基础上并行算法设计过程中,Harris 等[9] 提出 CUDA 并行原语设计思想。并行原语是基于 GPU 体系的数据并行处理集合,具有充分利用 GPU 硬件特性提高性能的优点,在数据库并行查询和数据挖掘分析算法中得到应用[10-11]。原语法封装了 CUDA 底层并行操作的实现,有效地分离并行数据计算逻辑与特定算法逻辑。在此采用基本原语法实现并行算法,首先对算法中原语操作定义如下。

定义 1　扫描。扫描的形式化定义为:

输入　$[c_0, c_1, c_2, \cdots]$。

输出　$[c_0, c_0 \oplus c_1, c_0 \oplus c_1 \oplus c_2, \cdots]$。

其中:$\oplus$ 为二元操作符号,如加减等。

定义 2　约简。约简是从一个输入数据集中计算得到一个值,形式化描述如下:

$$O[1] = \odot_{i=1}^{n} I[c_i]$$

其中:$\odot$ 表示约简函数,比如求和、最大、最小值等。对于约简原语采用多轮树状约简算法实现,充分利用数据时间局部性提高访存性能。若 CUDA 共享存储器空间为 $|M|$,则该原语需要执行 $\log_{|M|/r}[I]$ 步,其中 r 表示输入数据被分到 r 个线程块中,在第 i 步 $(0 \leqslant i \leqslant \log_{|M|/r}[I])$,线程 j 计算部分约简结果 $I_{[j\times 2i]}$ 和 $I_{[(j+1)\times 2i]}$。

定义 3　排序。排序是将一组无序数据按升序或降序排成有序数组。对于输入数组 I,输出数组 O 满足:$O[i] \leqslant O[j]$,$i \leqslant j$。本文算法中主要用到基数排序和双调排序,基数排序是非比较排序算法,算法时间复杂度为 $O(n)$;双调排序是一种与数据无关的排序算法,时间复杂度为 $O(n \text{ lb } n)$,当排序元素较少时效率高。

定义 4　过滤。过滤是从一组输入数据中筛选出满足特定条件的部分数据。形式化描述如下:

$$\{O[i] \mid 1 \leqslant i \leqslant m\} = \{I[i] \mid func(I[i]) = 1, 1 \leqslant i \leqslant n\}; m \leqslant n$$

定义 5　发散和聚集。分别是有索引读、写操作,定义如下:

$O[i] = I[L[i]]$

将输入数组 $L[i]$ 位置的元素读入到输出数组第 i 个位置;

$O[L[i]] = I[i]$

将输入数组第 i 个元素,写入输出数组第 $L[i]$ 位置处。

2.4　并行算法描述

根据线程调度模型和存储体系特点可改良成更优化的算法。

```
GPU_FN(θ)
    Global:
    TB: the number of threads in TB
    bid: index of TB
    tid: index of thread in TB[0,…,TB-1]
    Int Adj_Matrix[N][N];
    Int sub_Clusters[N][N];
    Int degree[N];
    Begin
```

```
1)    Initialize the sub_Clusters;
2)    While(cluster < θ)
3)      For each TB parallel do
4)        i = bid × TB + tid% k;
5)        degree[i] = prefixsum(Adj_Matrix[i], +, k);
6)      Endfor
7)      For each TB parallel do
8)        Call kernel calculating_deltaQ(sub_Clusters)
9)        ΔQ_{i,j} = reduce(max, ΔQ_{p,q})
10)     Endfor
11)     Mergeclustering(i, j);
12)     clusters --;
13)   Endwhile
    End
Kernel calculating_deltaQ(sub_Clusters)
    Int deltaQ[r × N]
    Int m_pq
    Begin
14)   i = bid × TB + tid;
15)   for each node_p in C_p
16)     for each node_q in C_q
17)       if (Adj_Matrix[node_p][node_q] = 1)
18)         m_pq++;
19)   deltaQ[i] = m_pq/m - (d_p + d_q)/2m^2
20)   synchrozine()
21)   if (tid = 0)
22)     Sort(deltaQ);
23)   ΔQ_{p,q} = deltaQ[0];
24)   Return ΔQ_{p,q};
    End
```

首先,初始化子聚类数组,每个节点为一个子簇(第 1)行),如图 1(a)所示。然后进入循环中执行,循环终止条件是当前复杂网络中簇的数量少于阈值 θ(第 2)行)。每次循环执行时以行为单位将邻接矩阵分为若干段,每个 TB 负责求一个段中节点的度。在该过程中以 Gather 原语读取存储在全局显存中的数组,用 Scatter 方式将计算结果存放在 *degree* 数组中(第 3)~6)行)。在并行求一个节点的度时采用前缀扫描求和原语,用 k 个线程并行处理邻接矩阵的一行数据(第 5)行)。分为两个阶段,每个阶段需要 lb N 步,N 是邻接矩阵一行数组大小,需要 $N-1$ 次加法和 $N-1$ 次交换,具体实现源于 CUDPP[9]。第 7)~10)行求 $\Delta Q_{p,q}$ 最大值,各线程块先并行计算每段中子簇与其他子簇的 $\Delta Q_{p,q}$,并求得 *TB* 内最大值 $\Delta Q_{p,q}$,各 TB 再通过 Reduce 求得全局最大值 $\Delta Q_{i,j}$(第 9)行);合并子簇 i, j 并更新子簇数组(第 11)行);进入下一次循环。

核函数 calculating_deltaQ 是算法中最耗时的过程,充分利用 GPU 的多线程并行计算 $\Delta \boldsymbol{Q}$ 矩阵。设每个线程块 *TB* 负责 r 个子簇 $C_1, C_2, \cdots, C_r$,求这 r 个子簇与其他 n 个子簇之间的 $\Delta \boldsymbol{Q}$,因为每个子簇最多分配 N 个线程能保证至少一个线程计算两个子簇之间的 $\Delta \boldsymbol{Q}$,所以线程块中线程数 $TB = r \times N$,其中 N 是复杂网络子簇的初始解。第 14) ~ 19)行的算法展示单个线程求解子簇 C_p 与 C_q 之间的 $\Delta Q_{p,q}$。首先根据线程序号确定该线程要处理子簇 C_p、C_q(第 14)行),然后遍历邻接矩阵求出子簇 C_p 与 C_q 合并后增加的连接数 m_{pq},再根据式(3)求出 $\Delta Q_{p,q}$ 并用 Scatter 将结果放入 *deltaQ* 数组中(第 19)行)。执行块内线程同步后,由线程中的 0 号线程调用 Sort 原语对 *deltaQ* 数组排序并将最大值输出。

2.5 负载均衡优化

由于 GPU 采用大量轻型线程并发执行代码,因此线程调度机制对 GPU 并行算法的性能有重要影响。持久型线程模型(Persistent Thread Model)是一种典型线程调度策略[12],在 2.4 节并行算法设计中采用该线程模型,在 CUDA 核函数的执行中发起充足线程去处理计算任务,直到全部任务执行完毕后才终止所有线程。将用该线程调度策略的并行算法简记为 FN_GP,然而 FN_GP 算法中存在负载不均衡现象。

在 FN_GP 算法中,子簇合并与 $\Delta \boldsymbol{Q}$ 矩阵计算属于重要核运算,具有多阶段流水线特征。但是随着网络簇合并数量的增加会出现子簇数组中数据倾斜的现象。如图 1(b)所示,子簇分布逐渐集中于网络簇矩阵的上部,而中下部分的子簇几乎处于无效状态。当 GPU 采用持久型线程模型调度多个线程并行处理网络簇矩阵时,必然要浪费线程。从图 2 可看出,分配处理黑色块的线程经过简单判断后就处于空闲状态。若子簇合并记为计算任务 T,则 $T = \{t_1, t_2, \cdots, t_n\}$,其中 t_i 是第 i 个子簇合并任务。各子任务的计算负载不同,$|t_i| \neq |t_j|$ $(i \neq j)$,并且 t_i 与 t_j 之间存在严格偏序关系,因此这类计算任务属于不规则负载的并行计算任务[13],需要合理划分数据负载来充分挖掘计算资源的利用率。

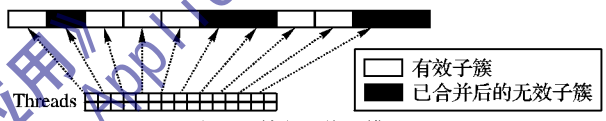

图 2 持久型线程模型

对持久型线程模型进行修改,实现提高线程利用率的目的,主要思想是对子簇数组的标志位进行过滤原语操作,过滤后全部为有效子簇,然后根据每次有效子簇的规律实时分配合理数量的线程来计算,如图 3 所示。调整线程调度策略的并行算法记为 FN_GU。然而由于 FN_GU 算法中增加 Filter、Gather 和 Scatter 3 种原语操作,数据收集浪费计算代价,因此在初始阶段引起较大性能损失。

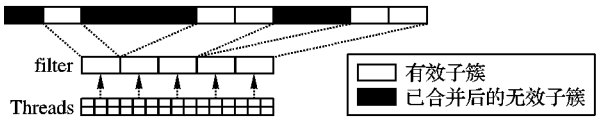

图 3 改良的持久型线程模型

为解决负载均衡和初始阶段的性能损失可同时使用两种线程调度策略。设定子聚类数的阈值,聚类初始阶段使用持久型线程模型调度线程;当子聚类数低于该阈值后才采用调整后的线程调度方案。将 FN_GPU 记为使用混合线程调度策略的并行算法,该算法的线程调度阈值需要通过实验方法确定,经过大量实验分析发现当子聚类数达到初始阶段的 23.7% 时调整线程调度策略最理想,因此将阈值设为23.7%。

2.6 存储优化

GPU 中存在层次存储体系,分别是全局存储器、纹理和常量存储器、片内共享存储器。其中全局存储器容量最大而存取延迟最大;而片内共享存储器为一个线程块所共享,容量为 16 KB,而存取速度接近寄存器;此外片内共享存储器可作为一个可实施软件管理的缓存使用,对于存在大量数据重用的算法可利用该存储区域进行优化。算术密度是一种衡量 GPU 通用计算效能的基本准则,算术密度是算术计算与访存操作的比率[14],算术密度越大 GPU 计算效能越高。例如,$y = x_1 \times x_2$,在计算过程中该表达式包括两次读,一次写存操

作及一次算术运算,算术密度为 1/3。若 GPU 的存储带宽为 22 Gbps,则计算速度仅为 7.3 Gbps。大多数计算中存在数据重用的问题,而缓存合理地优化数据访问有助于提高存取性能。例如矩阵 $M \times N$ 和 $N \times K$ 乘法运算,不考虑缓存的溢出命中率问题,则访存代价降为 N/K 和 N/M,无缓存的算术密度为 $2N-1/2N+1$,缓存容量无限条件下的算术密度为 $2N-1/N \times (1/M+1/K)+1$。如果 $M=K$ 且 N 足够大,则算术密度为 K,这种情况下带缓存的计算能够突破访存性能的瓶颈[15]。总之,GPU 并行算法设计中利用缓存优化访存机制是一种重要方法。

在复杂网络聚类算法中存在数据重用,在最耗时的网络簇合并计算过程中子簇节点的邻接矩阵需要多次读取,使用缓存能降低访存延迟。此外,数据重用并不是简单的静态方式,而是随着不同复杂网络实时调整的动态模式。其次,需要考虑 GPU 体系中缓存的硬件特性:1)GPU 与 CPU 的缓存不同,具有存储空间小和小工作集等特性;2)CUDA 存储体系中提供用户可管理的共享存储器可作为软件管理的缓存区域,然而软件管理缓存的复杂性及代价较高。

利用应用层缓存机制来解决动态数据重用,需要解决以下两个问题:第一,根据计算过程中的时间局部性确定缓存的数据;第二,确定缓存替换策略。在设计缓存管理机制时需要考虑计算复杂性对性能的影响,在此用最简单的缓存确定和替换策略。主要思想是将数据重用性最高的子簇节点邻接矩阵调入共享存储器中,以块为单位标记时间戳,该缓存块被调用一次后时间戳增加,以此记录缓存数据的生存时间。缓存数据的确定机制根据先使用先调入方式,最先使用的数据先进入缓存区域作为副本以供再次重用,该方法简单,计算效率高;缓存区替换采用最久不用置换策略,当需要调入新数据时将时间戳最小的数据块覆盖掉,这种替换策略符合时间局部性原理,具有较好的普适性和性价比。为便于讨论,增加缓存管理优化的并行算法简记为 FN_GPUM。

3　实验及分析

3.1　聚类精度比较

通过对比串、并行算法的聚类精度可验证并行算法的正确性。构造已知簇结构的随机网络为 $RN(C,s,d,pin)$,其中 C 为复杂网络中簇数目,s 为每个簇内节点数,d 为网络的平均度,pin 表示为簇内连接密度[2,8]。簇内连接密度大小反映复杂网络簇结构的特征,pin 越大簇结构越明显。在此采用 $RN(5,100,16,pin)$ 的 100 个随机网络,比较串、并行算法的聚类精度,如图 4 所示。

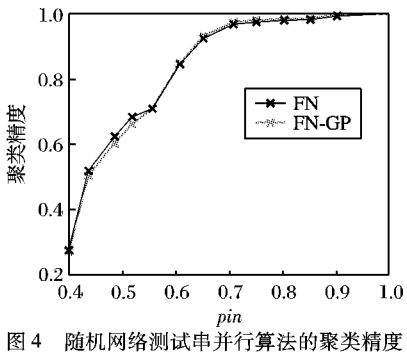

图 4　随机网络测试串并行算法的聚类精度

当复杂网络簇结构明显时(簇内连接密度 $pin > 0.8$)具有较高聚类精度;当簇结构不明显时($pin < 0.4$)聚类精度较低。从图 4 可见,在各种簇内密度情况下 GPU 并行算法的聚类精度与串行算法相似,因此通过已知簇结构的随机网络充分证明 GPU 并行算法的正确性。

3.2　计算速度比较

计算速度是评价复杂网络聚类算法的重要指标,串行快速聚类算法的时间复杂性为 $O(mn)$,其中 m 和 n 分别为网络的节点数和连接边数。

下面从实验角度提供串、并行算法的实际运行时间,作为比较性能的依据。以已知簇结构的随机网络 $RN(5,s,d,pin)$ 为测试数据,调整 s、d 的值构建不同规模的随机网络,其中有 $5s$ 个网络节点和 $5sd$ 个连接边,整体网络规模为 $5s(1+d)$。

由图 5 可见:串行算法实际运行时间与网络规模呈近似线性关系;并行算法在最大网络规模 7×10^4 处的执行时间为 195 s,整体并行计算比超过 10.5,特别在网络规模为区间 $[2.5 \times 10^4, 4 \times 10^4]$ 内并行加速效果最佳,稳定在 16.8 以上。当网络规模较小或较大以后,由于 GPU 与 CPU 之间需要数据交换,通信代价超过计算代价,导致并行效率下降。

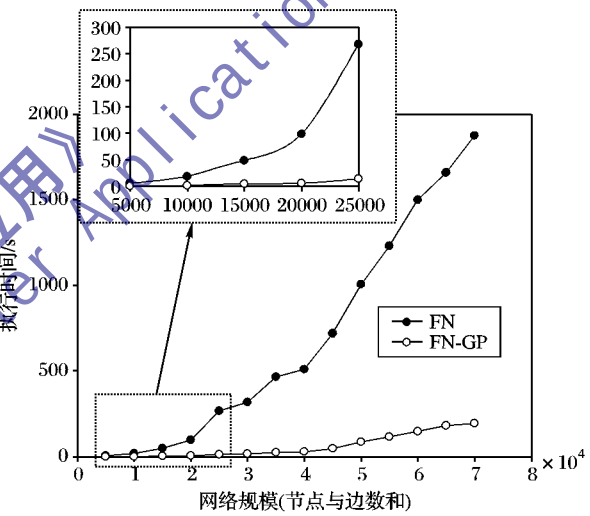

图 5　在不同规模网络下串、并行聚类算法实际执行时间

图 6 是在不同规模网络下 4 种并行算法的加速比对比。

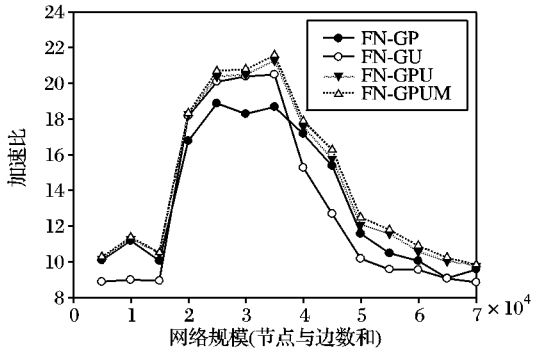

图 6　在不同规模网络下 4 种并行算法的加速比

前两种并行算法主要区别是线程调度策略不同,FN_GU 提高线程并发效率,能够减少计算功耗。在网络规模较小时,由于 FN_GU 算法需要准备计算数据,导致加速比低于 FN_GP;而在网络规模变大后,在区间 $[2.5 \times 10^4, 5 \times 10^4]$ 内,FN_GP 算法由于线程浪费,在处理较大规模矩阵时需要多轮加载,导致加速比低于 FN_GU。然而随着网络规模进一步增加,FN_GU 也需要多轮加载矩阵,这时两种算法加速比趋于相似。后两种并行算法属于优化版本,FN_GPU 结合两种线程调度算法的优点,其加速比一直优于前两种算法;因为增加

了缓存软管理策略,所以 FN_GPUM 算法加速性能好于 FN_GPU。

4 结语

在研究 GPU 并行快速复杂网络聚类基本算法之后,充分考虑线程调度模型对计算过程中负载均衡的影响,混合线程调度的算法在加速比和线程资源利用率方面高于未优化前的算法。此外,复杂网络聚类发现中重要数据的重用性也很突出,因此通过软件管理的缓存优化技术更进一步提高并行快速聚类算法的加速比。还一些问题需要进一步研究解决,比如优化聚类发现算法中压缩数据结构的存储问题,是处理超大规模复杂网络的研究方向;通过 GPU 集群解决大规模复杂网络聚类发现算法等。

参考文献:

[1] 唐晋韬,王挺,王戟. 适合复杂网络分析的最短路径近似算法[J]. 软件学报,2011,22(10):2279 - 2290.

[2] 杨博,刘大有,LIU JINMING,等. 复杂网络聚类方法[J]. 软件学报,2009,20(1):54 - 66.

[3] 骆志刚,丁凡,蒋晓舟,等. 复杂网络社团发现算法研究新进展[J]. 国防科技大学学报,2011,33(1):47 - 52.

[4] 何东晓,周栩,王佐,等. 复杂网络社区挖掘——基于聚类融合的遗传算法[J]. 自动化学报,2010,36(8):1160 - 1170.

[5] 黄发良,肖南峰. 基于线图与 PSO 的网络重叠社区发现[J]. 自动化学报,2011,37(9):1140 - 1144.

[6] 黄发良,肖南峰. 网络社区发现的粒子群优化算法[J]. 控制理论与应用,2011,28(9):1135 - 1141.

[7] 陈国强,王宇平. 基于极值优化模块密度的复杂网络社区检测[J]. 华中科技大学学报: 自然科学版,2011,39(4):82 - 85.

[8] NEWMAN M E J. Fast algorithm for detecting community structure in networks[J]. Physical Review E, 2004,69(6): 66 - 76.

[9] HARRIS M, OWENS J, SENGUPTA S, *et al*. CUDPP: CUDA data parallel primitives library[EB/OL]. [2010-10-10]. http://www.gpgpu.org/developer/cudapp/.

[10] HE B, LU M, YANG K, *et al*. Relational query co-processing on graphics processors[J]. ACM Transactions on Database System, 2009, 34(4): 1 - 39.

[11] 周国亮,陈红,李翠平,等. 基于图形处理器的并行方体计算[J]. 计算机学报,2010, 33(10):1788 - 1798.

[12] AILA T, LAINE S. Understanding the efficiency of ray traversal on GPUs[C]// HPG '09: Proceedings of the Conference on High Performance Graphics. New York: ACM Press, 2009: 145 - 149.

[13] TZENG S, PATNEY A, OWENS J D. Task management for irregular-paralle workloads on the GPU[C]// HPG '10: Proceedings of the Conference on High Performance. Piscataway, NJ: IEEE Press, 2010: 29 - 37.

[14] PHARR M, FERNANDO R. GPU Gems 2 [M]. Boston: Addison Wesley, 2005: 493 - 495.

[15] SILBERSTEIN M, SCHU A. GEIGER D, *et al*. Efficient computation of sum-products on GPUs through software-managed cache [C]// Proceedings of the 22nd Annual International Conference on Supercomputing. New York: ACM Press, 2008: 173 - 179.

(上接第 2440 页)

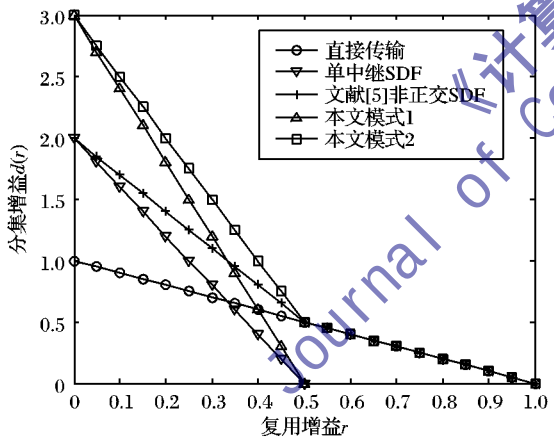

图 4 DMT 比较

4 结语

本文研究了两个中继之间存在屏蔽时,两路中继协作下的传输模式,得出了选择传输模式 2 在中断性能和 DMT 性能方面都有较大的提高。但对于存在 IRI 时的中断性能及消除方法,和多路中继及两路协作中继的选择策略需要进一步的研究。

参考文献:

[1] ZHENGLI-ZHONG, TSE D N C. Diversity and multiplexing: a fundamental tradeoff in multiple-antenna channels [J]. IEEE Transactions on Information Theory, 2003, 49(5): 1073 - 1096.

[2] 啜钢,刘振兴,王涛. Nakagami-m 信道下 DAF 选择中继的中断性能分析[J]. 北京邮电大学学报,2011, 34(1): 80 - 84.

[3] MA YI, TAFAZOLLI R, ZHANG YUAN-YUAN, *et al*. Adaptive modulation for opportunistic decode-and-forward relaying [J]. IEEE Transactions on Wireless Communications, 2011, 10(7): 2017 - 2022.

[4] LEE D, LEE J H. Outage probability of decode-and-forward opportunistic relaying in a multicell environment [J]. IEEE Transactions on Vehicular Technology, 2011, 60(4): 1925 - 1930.

[5] 张勇,徐友云,蔡跃明. 非正交选择译码转发协同[J]. 电子与信息学报,2008,30(9):2198 - 2202.

[6] SURAWEERA H A, SMITH P J, NALLANATHAN A, *et al*. Amplify-and-forward relaying with optimal and suboptimal transmit antenna selection [J]. IEEE Transactions on Wireless Communications, 2011, 10(6):1874 - 1885.

[7] HAGHIGHI A A, NAVAIE K. Outage analysis and diversity-multiplexing tradeoff bounds for opportunistic relaying coded cooperation and distributed space-time coding coded cooperation [J]. IEEE Transactions on Wireless Communications, 2010, 9(3): 1198 - 1206.

[8] RANKOV B, WITTNEBEN A. Spectral efficient protocols for half-duplex fading relay channels [J]. IEEE Journal on Selected Areas in Communications, 2007, 25(2):379 - 389.

[9] WICAKSANA H, TING S H, HO C K, *et al*. AF two-path half duplex relaying with inter-relay self interference cancellation: diversity analysis and its improvement [J]. IEEE Transactions on Wireless Communications, 2009, 8(9): 4720 - 4729.

[10] WICAKSANA H, TING S H, GUAN Y L, *et al*. Decode-and-forward two-path half-duplex relaying: diversity-multiplexing tradeoff analysis[J]. IEEE Transactions on Communications, 2011, 59(7): 1985 - 1994.

[11] TIAN FENG, ZHANG WEI, MA WING-KIN, *et al*. An effective distributed space-time code for two-path successive relay network [J]. IEEE Transactions on Communications, 2011, 59(8): 2254 - 2262.

[12] LUO C, GONG Y, ZHENG F. Full interference cancellation for two-path relay cooperative networks[J]. IEEE Transactions on Vehicular Technology, 2011, 60(1):343 - 347.