高技术通讯 2022 年 第32 卷 第9 期:903-913

doi:10.3772/j.issn.1002-0470.2022.09.003

基于顶点度数的图数据分区域重排序①

李 策② 章隆兵

(计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

(中国科学院大学计算机学院 北京 100190)

摘 要 图计算在机器学习、数据挖掘、网络安全等领域都有着重要应用。而图数据结构不规则且规模巨大,导致访存成为图应用运行时的瓶颈。由于图数据的顶点度数服从幂律分布,许多研究通过重排序图数据使高度数顶点连续储存在相邻位置,从而提升图数据被访问时的时间与空间局部性。然而重排序会破坏原始图数据中存在的群落结构,导致目前基于顶点度数信息的重排序算法在高结构性图数据集上无法产生性能提升。本文针对上述问题提出了一种新的保护图数据结构性的重排序算法,通过对自然图数据集中存在的群落结构进行研究,结合处理器访存结构特性,将图数据集合理划分成不同区域后进行重排序,保护其群落存储顺序以提高重排序后访存时的局部性。本文在通用处理器平台上,对6个不同结构性图数据集和3种图计算应用共18个测试点进行了验证,实验结果表明,该重排序方法相对于原始图数据集实现了平均18.86%的性能提升,相对于目前最优的基于顶点度数信息的轻量级重排序算法实现了平均11.3%的性能提升。

关键词 图数据重排序;顶点度数;访存局部性;幂律分布;群落结构

0 引 言

目前图计算正在金融、互联网、物流等领域得到广泛应用,相应的图算法种类繁多。由于图数据结构不规则,导致图计算访存的时间局部性和空间局部性较差,在运行时会产生大量访存缺失,大幅度降低了图计算在通用处理器上的运行效率[1-2]。

绝大部分图数据集的顶点度数服从幂律分布,即少量顶点连接大部分的边,大部分顶点连接少量的边。在图计算运算过程中,高度数顶点会被多次访问,低度数顶点占据较大存储空间而较少被访问[3],造成了不同度数顶点间时间局部性的不均衡。同时对于图数据,一个缓存行可以储存多个顶点的属性值,当度数差异较大的顶点储存在同一个缓存行时会降低访存的空间局部性[4],因而合理排列不同度数顶点可以有效挖掘图数据度数信息中潜在的访存局部性。

反映人类行为的真实图数据集中通常也包含众多群落结构[5-6],连续存储的顶点间往往属于同一个群落,彼此连接紧密,有较大概率拥有相同的邻顶点,运算时空间局部性高。同时,许多图计算框架下顶点的存储顺序代表顶点的执行顺序,相邻的顶点将被连续调度计算,同一群落的顶点被连续调度时,访问的源顶点大部分属于该群落,展现出较高的时间局部性。本文将含有较多群落的图数据集称为高结构性图数据集,反之则称为低结构性图数据集。简单地使用顶点度数信息对图数据进行重排序将会破坏原始图数据的群落结构,导致图计算性能降低。

为了在提升图数据局部性的同时,在一定程度上保护原始图数据的群落结构,本文提出了一种新

① 中国科学院战略性先导科技专项(C类)课题(XDC05020100)资助项目。

② 男,1992年生,博士生;研究方向:计算机系统结构,通用处理器设计,图计算加速;联系人,E-mail:lice@loongson.cn。

(收稿日期:2021-06-21)

的基于顶点度数信息的分区域图数据重排序算法。在保护图数据群落结构与挖掘顶点度数信息包含的局部性间达到了更好的平衡。

本文的主要贡献有以下 3 个方面。

(1)通过在并行图计算框架上对多种图数据集进行测试,发现了现有基于顶点度数信息的轻量级重排序算法在高结构性图数据集上的性能缺陷,并分析了产生该缺陷的原因。

(2)提出了一种分区域图数据重排序算法,通过研究图数据集中原有的群落特性,并结合通用处理器访存结构特点,对顶点数组合理划分区域后重排序,在保护数据集群落结构的同时提升了访存局部性。

(3)在不同的图计算应用和图数据集上对该重排序算法进行了验证,结果表明该算法相对于目前最优的基于顶点度数信息的重排序算法,实现了平均 11.3% 的性能提升,并适用于不同结构性的图数据集。

1 相关工作

为了挖掘图数据本身存在的局部性以提升图计算性能,近些年来许多工作围绕图数据预处理展开了研究。文献[7,8]将图数据划分成相同大小的区块,并保证每个区块都可以存储在片上缓存(cache)中。当图应用运行时,通过顺序处理每个区块,大幅度减少了应用内存访问的数量,从而提升了图计算的效率。但是上述划分需要对图算法进行改动,并且随着图数据集规模变大,区块数量也随之增加,不同区块间运算结果合并的开销逐渐抵消了带来的收益。而文献[9]则通过修改图数据储存结构,将顶点信息和结构信息联合储存后进行数据分块,保证每个区块的数据都可以存储在片上缓存中,从而使得图应用在计算过程中读写图数据时具有良好的空间局部性。但该方法需要占用较大的存储空间,且适用的图应用类型有限。

为了利用真实图数据集中存在的群落结构以提升图计算访存的时空局部性,文献[10]分析了任意两个顶点间的关联度,认为关联度较高的顶点有更大概率属于同一个群落。通过将较高关联度的顶点存储在内存中的相邻位置,该重排序算法显著提高了图计算访存时的局部性。但由于该算法时间复杂度过高,产生的预处理开销超出了重排序收益。与此同时,由于相互连接的顶点间有更大概率属于相同群落,文献[11-14]通过对图数据集进行深度优先遍历或广度优先遍历(breath first search, BFS)来挖掘其群落结构中存在的局部性。文献[7]则首先通过宽度优先搜索将图分成限定大小的部分,之后再以深度优先搜索的顺序对各部分顶点进行重排序。而文献[14]提出了一种异步图计算处理框架,以深度优先顺序将图数据集划分成不同部分,并按照该顺序对图顶点进行计算调度,大幅度提升了图计算访存时的时间局部性。但由于没有对顶点进行储存上的重排序,访存的空间局部性未获得提升。虽然上述几种预处理方案能较好地利用图数据中的群落结构,但对图数据集进行深度优先或广度优先遍历所需的开销也不可忽略。针对上述问题,文献[15]提出了一种硬件解决方案,通过添加硬件结构并修改算法,实现了运行时对图顶点的深度优先调度,在充分提升访存时间局部性的同时,消除了图数据预处理开销。然而该方案需要添加额外的硬件电路,同时无法应用于 BFS 和单源最短路径(single-source shortest path, SSSP)等有固定调度顺序的图算法。

基于减少预处理开销以提升重排序方案可行性的出发点,目前许多研究提出了基于顶点度数信息的重排序算法[3,7,16],通过将高度数顶点连续存储在相邻位置,减少了存储高度数顶点所需的缓存行数量,提升了访存时的时空局部性。但同时此类方法也会破坏原始图数据中群落结构存储的密集程度,本文接下来将简要介绍该类重排序算法的研究现状。

2 图数据重排序研究背景

本节主要对图计算的基础知识和基于顶点度数信息的图数据重排序算法进行介绍和分析。

2.1 图数据存储格式

压缩稀疏行存储(compress sparse row, CSR)是

一种被广泛使用的图数据存储格式,使用该格式存储图数据时空间利用率较高。图 1 为 CSR 存储模式示意图,CSR 格式使用两个数组存储图数据的空间结构。其中偏移数组存储每个顶点对应的第一条边在边数组中的地址,边数组则按照顶点顺序存储每个顶点做为目的顶点时,连接的所有边的源顶点序号。而图应用运算时产生的数据则存储在顶点数组中,不同的图应用对应的顶点数据内容有所不同,例如在 PageRank 算法中顶点数组存储的是每个顶点对应的网页权重。而 SSSP 中存储的则是路径长度。同时在某些需要权重的算法中,每条边的权值也会被存储在边数组中,便于运行时读取。

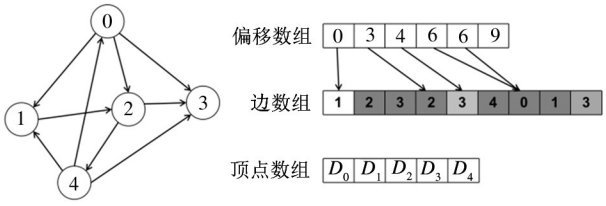

图 1　CSR 存储格式

2.2　图计算访存特点

目前许多图计算框架[17-19]都采用以顶点为中心(vertex-centric)的计算模式。在该计算模式下,对每个顶点属性值进行计算时,需要读取该顶点所有相邻顶点的属性值。当使用 CSR 格式存储图数据时,以 PageRank 算法为例,对应的访存过程如下:(1) 访问偏移数组读取计算顶点第一条边的偏移地址;(2) 在边数组中获取该边的源顶点编号;(3) 读取顶点数组中该源顶点对应的属性值;(4) 对图中所有活跃顶点重复上述操作完成计算。

通过上述访存过程可知,每次计算过程中偏移数组和边数组会被顺序遍历一次,有较高的空间局部性,无时间局部性。但对顶点数组的访问是随机且不规则的,因为每条边的源顶点编号无确定规律,所以访存的时空局部性较低,是图计算运行时的主要瓶颈。图数据重排序便是通过重新排列顶点的储存顺序,来挖掘顶点数组潜在的访存局部性。

2.3　基于顶点度数的图数据重排序

基于顶点度数的图数据集重排序主要利用了图数据集幂律分布的特点,通过将高度数顶点连续储存在相邻位置,来提升访存的时空局部性。由于顶点间度数差异较大,每个顶点被访问的次数和度数成正比,同时一个 64 字节的缓存行内可以同时储存 4 ~ 16 个顶点。如果将高度数顶点连续储存在一个缓存行内,可以减少高度数顶点占用的总缓存行数,达到提升访存空间局部性的目的。同时在以顶点为中心的计算模式中,当程序按照顶点顺序对图数据进行遍历计算时,高度数顶点间有更大概率拥有更多的共同邻顶点,访存的时间局部性较高。

但由于真实的图数据集具有群落结构,部分顶点间连接更加紧密。在以顶点为中心的计算模式下,属于同一群落的顶点有更大可能被同时访问。且由于同一群落顶点间拥有相邻顶点的数量相对较多,对相同群落内的顶点连续调度进行计算时,会提升访存的时间局部性。在结构性较好的图数据集中,属于同一群落的顶点往往会存储在相邻位置,本身便具有较好的空间局部性。而顶点度数信息不能反映群落结构特性,所以根据顶点度数信息进行重排序时容易打破原始图数据集中群落结构的存储顺序,造成图计算性能的降低。

2.4　重排序算法对比

为了在提升图数据访存局部性的同时,减少对图数据中群落结构的破坏,许多基于顶点度数信息的重排序算法做出了相应改进。本文主要对几种经典的基于顶点度数信息的重排序算法进行了比较,算法的排序方案如图 2 所示。对于其他基于图结构信息的重排序算法如文献[10,20]等,虽然实现了较高的性能提升,但预处理开销过大。综合来看时,往往不能获得收益,因而不在本文考虑范围。

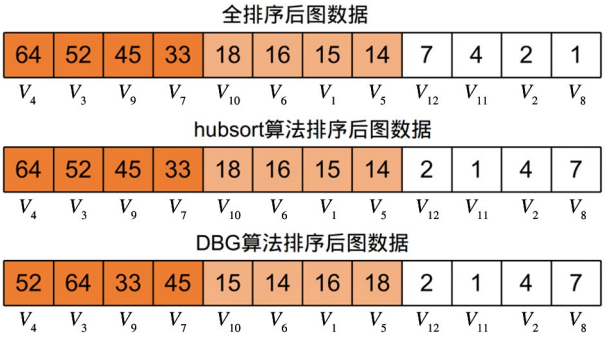

图 2　不同重排序算法排序示例

全排序算法按照顶点的度数值对所有顶点进行全排序,该算法可以最大程度地减少储存高度数顶点所需的缓存行数量,从顶点度数角度最大限度地提升了图数据存储的空间局部性。但是对图数据中群落结构的破坏相对较大。为了保护图数据中的群落结构,增强图数据重排序算法在高结构性图数据集上的性能表现,hubsort 算法[7]将图数据的平均度数设定成划分阈值,根据该阈值将图顶点储存成两部分,对高于度数阈值的顶点进行全排序,而低于度数阈值部分的顶点不进行排序。该方法在一定程度上保护了低度数顶点群落结构的存储密度,但对高度数顶点的结构破坏依然较大。DBG(degree-based grouping)算法[3]在 hubsort 算法的基础上进一步改进,提出了多阈值划分,利用图数据幂律分布的特性,设定了一系列呈指数增长的阈值,将顶点数组按照阈值范围划分成不同存储部分。同一部分的顶点间不再进行排序,缩短重排序时间的同时更好地保护了图数据的原有结构。该算法是目前最优的基于顶点度数信息的轻量级图数据重排序算法。DBG 算法在低结构性图数据集上有较好的性能表现,但是对于高结构性的图数据集,DBG 算法往往不能实现性能提升。表 1 展示了基于顶点度数信息的重排序算法在高结构性图数据集上,相对于原始图数据集的性能表现。测试算法为 PageRank。

表 1　不同排序算法在高结构性图数据上的性能表现

算法	arabic	uk	sk	webbase
Sort	-83.2 %	-84.1 %	-5.9 %	-67.1 %
Hubsort	-70.5 %	-68.4 %	0.8 %	-81.4 %
DBG	-9.2 %	-16.5 %	10.7 %	-23.7 %

2.5　基于顶点度数的重排序算法的性能缺陷分析

从表 1 的结果可以看出,基于顶点度数信息的重排序算法在许多高结构性图数据集上会产生严重的性能下降,相对来看 DBG 算法产生的性能降低最小。这是由于 DBG 算法通过多阈值的顶点度数划分,对图数据原有的群落存储顺序实现了较好的保护。但该算法仍然存在性能缺陷,因而可以通过研究 DBG 算法产生该缺陷的原因,来提升基于顶点度数信息类重排序算法在高结构性图数据集上的性能表现。

为了研究 DBG 算法的缺点,本文提出了一个新的衡量图数据存储时群落结构密度的标准:群落储存密度,即某一区域内顶点的邻顶点隶属于该区域的数量占该区域顶点的所有邻顶点数量的比例。因为同一群落顶点的邻顶点有更大概率属于该群落,所以上述比例越大说明同一群落的顶点存储密度越大,此时访问图数据的时空局部性越好,而上述比例越小则说明同一群落的顶点存储较为稀疏,访存时空局部性差。图 3 是经过 DBG 排序后图数据的群落存储密度与原始图数据群落存储密度的对比,其中 ORG 代表原始图数据。实验选取的区域大小是 20 480,代表每个测量区域含有20 480个顶点。可以看出,DBG 排序降低了图数据中群落的存储密度,导致了同一群落的顶点稀疏地分布在内存中。因为 DBG 算法只保证了同一度数范围内的顶点的相对存储顺序不被改变,但大幅度改变了不同度数范围顶点间的排列顺序,使紧密存储的群落顶点分布在不同度数区域,降低了群落顶点连续调度带来的时间局部性的提升,以及紧密存储带来的空间局部性的提升,最终造成了图计算的性能降低。同样的问题也存在于其他基于顶点度数信息的重排序算法中。

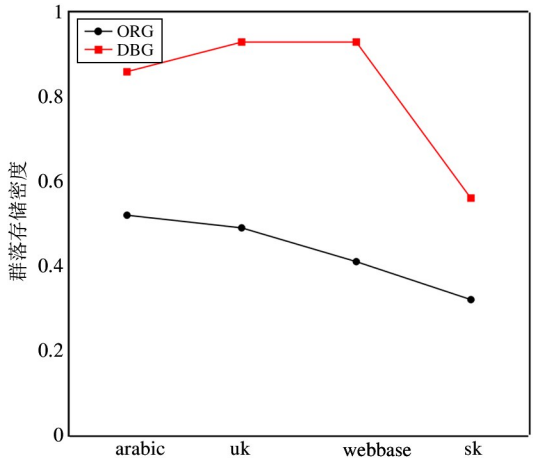

图 3　群落存储密度对比

3　分区域图数据重排序

针对 DBG 算法的性能缺陷,本文提出了一种分区域图数据重排序方案。首先按照合理规模将图顶

点分割成不同的区域,在每个区域内部进行基于顶点度数的重排序,该方案从顶点调度顺序的角度更好地保护了原始图数据中存储的群落结构,在高结构性的图数据集上有着更好的性能表现。

3.1　分区域重排序算法

之前的实验和分析结果表明,目前基于顶点度数的图数据重排序算法在许多图数据集上会产生性能下降。根本原因是该重排序方案对高结构性图数据集存储时的群落密集度破坏较大,导致访存时的时空局部性降低。为了在提升访存局部性与保护原始图数据群落结构间达到更好的平衡,进一步保护原始图数据集存储时的群落密集度,本文提出了分区域图数据重排序算法,流程如算法 1 所示。

算法 1　区域重排序算法

在 $\boldsymbol{G}(V,E)$ 中,V 代表图中顶点数量,E 代表图中边的数量

1: **For** v from 1 to V

　　if $S \times k \leqslant v < S \times (k+1)$

　　$v \in Area_k$

其中 S 为每个区域的规模

2: **For** every v in $Area_k$

　　if $Q_n \leqslant D[v] < Q_{n+1}$

　　$v \in Group_{kn}$

其中 $D[v]$ 代表每个顶点的度数, Q_n 为度数划分的阈值

3: **For** every $Area_k$

　　id: $= S \times k$;

　　For every $Group_{kn}$ from 1 to N

　　For every $v \in Group_{kn}$

　　$New[v]$: = id + +

其中 $New[v]$ 代表顶点 v 的新编号

算法首先设定一个固定区域规模,之后根据该规模将顶点数组顺序分割成不同的区域,通过在每个区域内部设定一系列度数阈值来对顶点进行重排序。区域划分降低了相同区域内顶点的调度顺序的改变,而隶属于每个区域的群落顶点间储存密集度不会变化。同时也减少了分布在相邻区域的群落顶点间存储顺序的变化,从而保护了原始图数据集中群落的存储结构。之后通过在每个区域内部基于顶点度数范围的重排序,减少了高度数顶点存储所需的缓存行数量,保证了重排序算法在低结构性图数据集上的性能提升。因而分区域重排序方案可以同时适用于不同结构性的原始图数据集。图 4 是区域重排序算法的一个具体示例,示例中区域大小为 4,即每 4 个顶点为一个区域。在每个区域内部,通过设定度数阈值 16 和 32 将图数据重排序成 3 个部分,每部分顶点间不进行重排序。

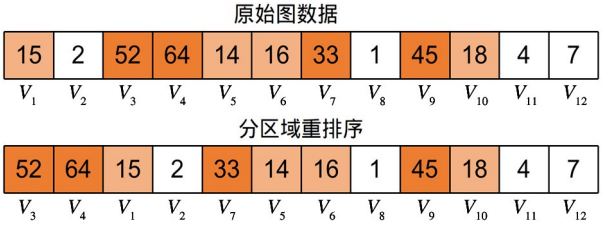

图 4　区域重排序算法示例

3.2　区域规模选取

区域规模的选取在本文的算法中至关重要。区域内包含的顶点数量过少,则不能挖掘顶点度数信息中包含的图数据潜在的访存局部性,顶点数量过多则会导致图应用运行时计算同一区域顶点需要访问的顶点数量超出核内私有缓存的容量,需要到共享缓存或内存中读取数据。增加了处理器的访存延迟,降低了图计算的运行性能,因而区域规模应选取在一个合理的范围。

针对不同配置的通用处理器,本文提出了一种区域规模选取的参考模型,设区域规模大小为 N,则 N 的取值范围为

$$\max(A/C,\ M) \leqslant N \leqslant (A+B)/C$$

其中 A 为处理器中一级 cache 的容量, B 为处理器中二级 cache 的容量,一级 cache 和二级 cache 为包含关系。C 为图数据集中每个顶点数据的大小, M 为真实图数据中群落内平均包含的顶点数量。

由于一级 cache 能存储的顶点数量为 A/C, 二级 cache 可以存储的顶点数量为 B/C, 而图计算运行时访问的顶点数据如果存储在一级或者二级 cache 内时,访问逻辑链路控制(logic link control, LLC)和内存的次数便随之降低,所以区域内顶点数量不应超过一级 cache 与二级 cache 所能容纳的顶点数 $(A+B)/C$。为了保证每个区域内有足够数量的高度数顶点,区域规模不应小于一级 cache 容纳的顶点数量A/C。而区域规模选取时还需要考虑真

实图数据集中群落的大小，如果将一个群落划分在不同区域，重排序会导致该群落顶点间调度顺序相距较远。因而区域规模不应小于图数据集中群落平均包含的顶点数 M，结合对处理器配置的分析，区域规模 N 应大于 M 与 A/C 中的最大值，而当 M 大于二级缓存容量时，N 的值设为 $(A + B)/C$。

为了确定真实图数据集中群落的平均大小，本文对各类图数据在不同区域规模下的群落储存密度进行了统计，实验结果如图 5 所示。对于低结构性图数据集 twitter，群落存储密度一直较低，且随区域规模增加无明显变化。而在高结构性图数据集上群落存储密度初始时随区域规模变化快速增长，当区域规模到达一定数值时，群落存储密度开始缓慢变化并最终接近 1。即在某个数值时，高结构图数据集对同一区域内顶点运算时访问的源顶点基本落在了本区域内，此时该数值可认定为图数据集中群落的平均大小，在图 5 中该数值近似为 4096。

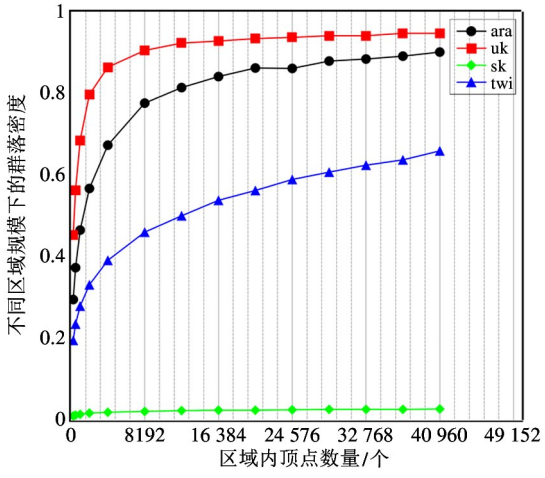

图 5　图数据集群落存储密度变化趋势

通过上述实验与分析，对于不同配置处理器和真实图数据集，本文算法选取的区域规模 N 的大小范围为 $\max(A/C,\ 4096) \leqslant N \leqslant (A + B)/C$。当 $(A + B)/C < 4096$ 时 N 值为 4096，且在上述区域范围内对 N 进行探索时以 4096 为递增单位。

3.3　算法总结

本文提出了一种新的基于顶点度数信息的重排序算法，相对于 DBG 算法增加了新的设计探索参数即区域规模，同时本文算法中也包含 DBG 中的探索变量即度数范围。DBG 和原有的其他基于顶点度数的重排序算法，可以通过修改规模参数在本文的算法下实现，当区域规模设置为图中总的顶点数时，本方案实现的是 DBG 排序算法，而当区域大小设为 1 时，本方案对应的是原始图数据集的存储顺序。

4　实验结果与分析

本节将介绍实验所用数据和平台，展示最终的实验结果并进行分析。

4.1　测试输入

为了验证本文提出的重排序算法对图计算的性能提升，本文选取了 3 种常见的图计算算法。如表 2所示，其中 PageRank 算法对全部顶点进行迭代计算，运行时间最长，重排序收益最高；Radii 算法和 CC 算法中只有部分顶点参与运算。

表 2　图算法描述

图算法	描述
PageRank(PR)	一种根据相邻顶点信息来计算顶点排名的迭代算法，用于网页排名的计算[21]
Radii Estimation(Radii)	一种通过并行 BFS 计算图中每个顶点半径的算法[22]
Conn. Components(CC)	一种计算图中所有的连通分量的算法，连通分量是图的一个子图，子图中任意两点之间均存在可达路径[23]

表 3 列出了本文选取的 6 个不同类型的真实图数据集，主要来自于网页和社交网络。顶点数据的规模超过实验服务器 LLC 的容量，能较好地体现重排序算法对内存带宽利用率的提升。图数据的直径

表 3　图数据集描述

图数据集	顶点数	边数	结构性	平均度数
indochina[24]	7×10^6	194×10^6	高	25
arabic[24]	22×10^6	640×10^6	高	29
pld[3]	43×10^6	623×10^6	低	14
sk-2005[24]	51×10^6	1949×10^6	高	38
it[24]	41×10^6	1150×10^6	高	27
twitter[25]	62×10^6	1648×10^6	低	27

范围为 14 ~ 38,聚集系数为 0.06 ~ 0.55,其中聚集系数代表了图数据集群落结构的密集程度。数据集 pld 和 twitter 为低结构性图数据集,其余数据集初始结构性较高。实验选取的 6 个真实图数据集具有广泛的代表性,能反映现实中大部分图数据集的特点,将上述图数据集作为测试输入可以验证本文提出的重排序算法的普适性。

4.2　测试平台

本文测试的源代码来自目前被广泛使用的图计算框架 ligra[11]。重排序算法的实现与重排序时间的计算参考了文献[3]。使用的编译器版本为 g ++ 7.5,采用-O3 优化,操作系统版本为 ubuntu-16.04,内核版本为 linux-4.4.0-189-generic。并行实现由两部分组成,重排序算法采用 OpenMP 实现并行,图算法采用 Cilk plus 实现并行。实验中发现,采用 OpenMP 实现图算法并行时,程序不能充分利用 CPU 资源。服务器由两路 NUMA 节点构成,程序运行时使用 numactl -i all 选项优化内存分配。本文的实验服务器配置信息如表 4 所示,NUMA 节点由双路的 intel E5-2620 v4 组成,每路中包括 8 个核心和 20 MB 的 LLC。

表 4　服务器配置信息

参数	描述
操作系统	Ubuntu-16.04
处理器	Intel E5-2620 v4
主频/GHz	2.1
L1 dcache/kB	32
L2 cache/kB	256
LLC/MB	20
Memory/GB	64
核数	32

为了更准确地评估重排序后的程序运行时间,每个应用与数据集的测试组合运行了 11 次。排除对 cache 进行预热的第 1 次运行的时间,计算出后续 10 次运行时间的算术平均值作为最后的测试结果。在实验运行选项下服务器 CPU 资源利用率达到最高,多次重复实验结果误差在 2% 以内。

4.3　区域大小探索

为了探索区域划分规模对性能提升的影响,结合第 3 节的区域规模分析模型,本文对所有合理数值进行了测试。实验所用服务器中一级 cache 大小为 32 kB,顶点大小为 8 字节,一级 cache 能容纳的顶点数量是 4096 个,二级 cache 大小为 256 kB 能容纳的顶点数量为 32 768 个。根据第 3 节的模型,规模大小 N 在 4096 ~ 36 864 之间,测试增量为 4096,需要对 9 个数值的区域规模进行设计探索。最终结果如图 6 所示,当区域内顶点数为 20 480 个时,本文的算法相对于 DBG 算法达到最优的性能提升。

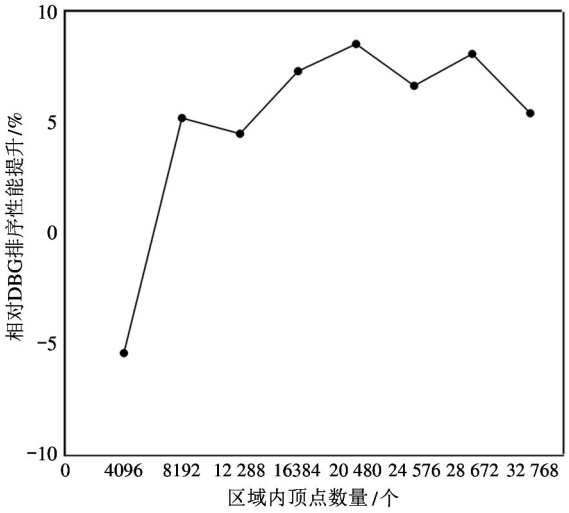

图 6　不同区域规模下的性能提升

需要说明的是,当 N 等于 4096 时,本文算法相对于 DBG 算法产生了一定的性能降低。这是由于区域过小时,容纳的高度数顶点较少,无法产生较高的性能收益,但是相对于原始图数据依然有性能提升。因而 N 值的选取只需在本文提出的范围内即可,无需花费时间选取最优值。

4.4　运行时间对比

由于本文是针对预处理开销较低的基于顶点度数信息的图数据重排序算法进行改进,而 DBG 算法相对于其他同类排序算法[7,16]预处理时间短,对群落结构保护最好实现性能提升最高。所以本文只需将分区域排序算法与 DBG 算法进行对比,便可以验证本文算法相对于其他同类算法,是否能更好地保护群落结构,在高结构性图数据集上实现性能提升。其中 DBG 排序的实现与文献[3]中一致,度数范围为 $[32A, +\infty)$、$[16A,32A)$、$[8A,16A)$、$[4A,8A)$、$[2A,4A)$、$[A,2A)$、$[A/2,A)$ 和 $[0,A/2)$,共 8 个区

域,其中 A 代表每个图数据集的平均度数。不同的是 DBG 算法运行时使用了 OpenMP 实现并行,而本实验中采用了 Cilk plus 实现并行,可以提升运行速度。图 7 是不考虑重排序时间的测试结果,其中数据集 pld 和 twitter 来自文献[3]中使用的低结构图数据集。在这 2 个数据集上,本文的实现方案相对于 DBG 方案有一定的性能下降。但是相对于原始图数据集,本文提出的重排序算法,依然可以产生较高的性能提升,证明分区域重排序算法可以在低结构数据集上取得较好的性能表现。同时在高结构性图数据集上,为了验证区域规模选取模型的正确性,实验时选取了两个不同于之前测试的高结构性图数据集 indochina 和 it。结果表明本文的方案相对于 DBG 算法皆实现了不同程度的提升,最高可以达到 62%。综合来看,分区域图数据重排序算法在不同结构性的图数据集上都有较好的性能表现。相比于 DBG 排序算法,本文方案在高结构性图数据集上实现了性能提升,具有更高的普适性。

图 7　两种重排序算法相对于原始图数据的性能提升

4.5　重排序时间对比

DBG 重排序并行实现时,使用的是 OpenMP 静态调度。分区域重排序使用 OpenMP 动态调度实现并行,根据区域规模确定动态调度划分的方式,充分利用 CPU 资源。图 8 是 2 种算法重排序时间对比。结果表明,2 种算法重排序时间基本一致,相差不超出 0.2 s。将重排序时间计入到总的运行时间后,本文的排序方案依然优于 DBG 算法。同时相对于原始图数据集,由于 PageRank 算法运行时需要多次迭代,预处理带来的额外开销完全可以被重排序的收益抵消。而对原始图数据集一次重排序后,可以运行多种图算法,所以将预处理开销计入总的运行时间后,本文提出的算法依然可以产生较高的性能收益。

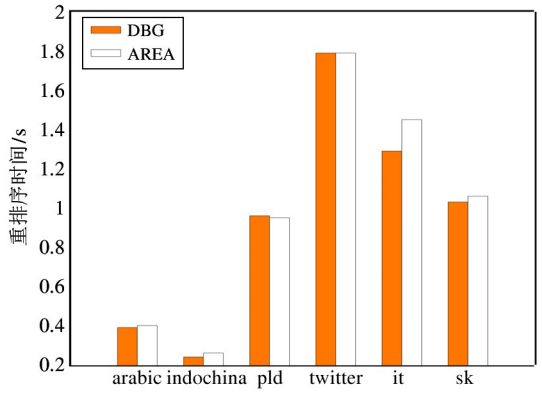

图 8　两种重排序算法预处理时间对比

4.6　性能分析

本文通过 perf 性能分析工具,抓取了不同图数据集运行 PageRank 算法时 L1 cache 和 LLC 的缺失率。分析重排序算法改进实现计算性能提升的原因,验证算法设计的正确性。

L1 cache 的每千条指令未命中数(miss per kilo instructions,MPKI)统计结果如图 9 所示。实验结果显示,低结构性图数据集 pld 和 twitter 的 MPKI 明显高于其他图数据集,因为拥有较好群落结构的图数据集访存局部性高,产生的缓存缺失数量少。相比于 DBG 算法,本文提出的算法(AREA)对于 L1

cache 缺失没有明显提升。因为当划分的区域大小为 20 480 时,每个区域运算时访问边的源顶点数量超过 L1 cache 的容量,大部分访存命中在二级 cache 与 LLC 中,所以分区域重排序不会显著减少 L1 cache 缺失。

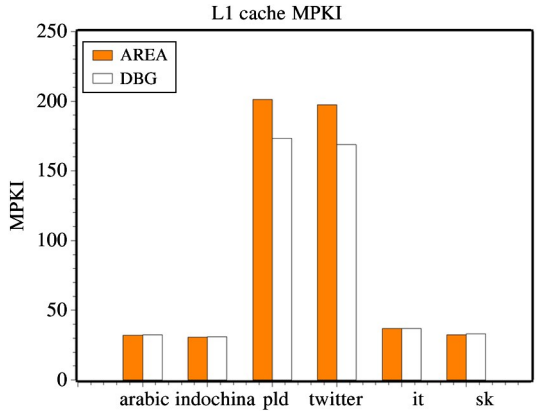

图 9　L1 cache MPKI 对比

LLC MPKI 统计结果如图 10 所示。图中没有列出低结构性数据集 pld 和 twitter 的统计结果,二者的统计结果与其他数据集相比高出一个数量级。在这 2 个数据集上分区域排序相对于 DBG 算法 MPKI 分别增加了 4% 到 10%,显著低于 L1 cache 的 MPKI 增加(33%,25%),表明在低结构性图数据上分区域排序相对 DBG 排序算法的性能降低主要来自于 L1 cache 缺失率的增加。而如图 10 所示,在高结构性图数据集上,分区域重排序显著降低了 LLC 的缺失数量。因为分区域排序没有改变群落的存储密度,但缩减了高度数顶点的存储容量。群落内顶点

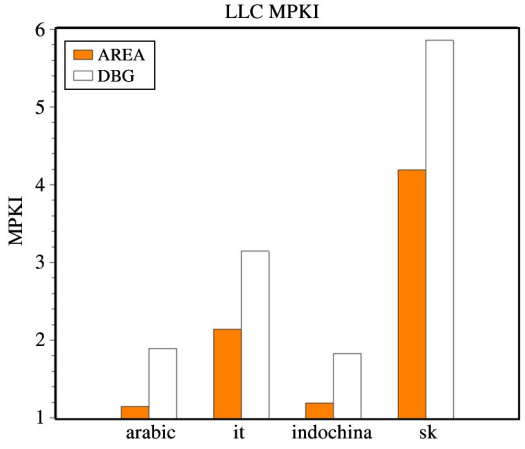

图 10　LLC MPKI 对比

进行计算时访问的数据有较大概率命中在片上 cache 中,降低了访问片外 DRAM 所需的次数。

L1 cache 和 LLC MPKI 数据的统计结果表明,本文提出的算法在保护原始图数据群落结构存储密度的同时,提升了高度数顶点的访存局部性。使相同群落内计算时需要访问的顶点数据基本存储在 LLC 内,缩减了访问片外 DRAM 的数量,提升了高结构性图数据集运行图计算时的性能表现。对于低结构性图数据集,分区域重排序算法通过改变区域内顶点的局部性,同样减少了访问 LLC 的缺失率,相对于原始图数据集实现了较高的性能提升。

5　结　论

本文通过对基于顶点度数信息的不同重排序算法的研究,发现了该类重排序算法在高结构性图数据集上的性能缺陷。为了更好地保护图数据集原有的群落结构,本文从维护图顶点群落存储密度的角度出发,提出了一种新的分区域图数据重排序算法。通过对顶点数组的区域划分,减少了由于重排序造成的顶点间调度顺序的改变,维护了原有图数据集群落存储密度,提升该类重排序算法在高结构性图数据集上的性能表现。在不同类型的图数据集上的实验结果表明,该重排序算法相对于目前最优的基于顶点度数信息类重排序算法 DBG 实现了平均 11.3% 的性能提升,且适用于不同结构性的原始图数据集。

参考文献

[1] 严明玉, 李涵, 邓磊, 等. 图计算加速架构综述[J]. 计算机研究与发展, 2021, 58(4): 862

[2] 张承龙, 曹华伟, 王国波, 等. 面向高通量计算机的图算法优化技术[J]. 计算机研究与发展, 2020, 57(6): 1152

[3] FALDU P, DIAMOND J, GROT B. A closer look at lightweight graph reordering[C]//2019 IEEE International Symposium on Workload Characterization (IISWC), Orlando, USA, 2019:1-13

[4] BALAJI V, CRAGO N, JALEEL A, et al. P-OPT: practical optimal cache replacement for graph analytics[C]//

2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Korea, 2021: 668-681

[5] GIRVAN M, NEWMAN M E J. Community structure in social and biological networks[J]. *Proceedings of the National Academy of Sciences*, 2002, 99(12): 7821-7826

[6] LESKOVEC J, LANG K J, DASGUPTA A, et al. Statistical properties of community structure in large social and information networks[C]// Proceedings of the 17th International Conference on World Wide Web, New York, USA, 2008: 695-704

[7] ZHANG Y, KIRIANSKY V, MENDIS C, et al. Making caches work for graph analytics[C]//2017 IEEE International Conference on Big Data, Boston, USA, 2017: 293-302

[8] NISHTALA R, VUDUC R W, DEMMEL J W, et al. When cache blocking of sparse matrix vector multiply works and why[J]. *Applicable Algebra in Engineering, Communication and Computing*, 2007, 18(3): 297-311

[9] BEAMER S, ASANOVIC K, PATTERSON D. Reducing pagerank communication via propagation blocking[C]// 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Orlando, USA, 2017: 820-831

[10] WEI H, YU J X, LU C, et al. Speedup graph processing by graph ordering[C]// Proceedings of the 2016 International Conference on Management of Data, San Francisco, USA, 2016: 1813-1828

[11] BANERJEE J, KIM W, KIM S J, et al. Clustering a DAG for CAD Databases[J]. *IEEE Transactions on Software Engineering*, 1988, 14(11): 1684-1699

[12] CUTHILL E, MCKEE J. Reducing the bandwidth of sparse symmetric matrices[C]// Proceedings of the 24th National Conference, Washington, USA, 1969: 157-172

[13] YUAN P, XIE C, LIU L, et al. PathGraph: a path centric graph processing system[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2016, 27(10): 2998-3012

[14] ZHANG Y, LIAO X, JIN H, et al. FBSGraph: accelerating asynchronous graph processing via forward and backward sweeping[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 30(5): 895-907

[15] MUKKARA A, BECKMANN N, ABEYDEERA M, et al. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling[C]// 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 2018: 1-14

[16] BALAJI V, LUCIA B. When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs[C]// 2018 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, USA, 2018: 203-214

[17] SHUN J, BLELLOCH G E. Ligra: a lightweight graph processing framework for shared memory[C]// Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Shenzhen, China, 2013: 135-146

[18] BEAMER S, ASANOVICK, PATTERSON D. The GAP benchmark suite[EB/OL]. http://arxiv.org/abs/1508.03619: arXiv, (2015-08-14), [2021-04-21]

[19] SUNDARAM N, SATISH N R, PATWARY M M A, et al. Graphmat: high performance graph analytics made productive [EB/OL]. http://arxiv.org/abs/1503.07241: arXiv, (2015-03-25), [2021-04-21]

[20] HUANG B, LIU Z, WU K. Structure preserved graph reordering for fast graph processing without the pain[C]// 2020 IEEE 22nd International Conference on High Performance Computing and Communications, Yanuca Island, Fiji, 2020: 44-51

[21] PAGE L, BRIN S, MOTWANI R, et al. The PageRank Citation Ranking: Bringing Order to The Web[R]. Santa Clara County: Stanford InfoLab, 1999

[22] MAGNIEN C, LATAPY M, HABIB M. Fast computation of empirically tight bounds for the diameter of massive graphs[J]. *Journal of Experimental Algorithmics (JEA)*, 2009, 13: 1.10-1.9

[23] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to Algorithms[M]. Cambridge: MIT Press, 2009: 20-23

[24] DAVIS T A, HU Y. The University of Florida sparse matrix collection[J]. *ACM Transactions on Mathematical Software (TOMS)*, 2011, 38(1): 1-25

[25] KWAK H, LEE C, PARK H, et al. What is Twitter, a social network or a news media? [C]// Proceedings of the 19th International Conference on World Wide Web, Raleigh, USA, 2010: 591-600

Graph reordering in area according to vertex degree

LI Ce, ZHANG Longbing
(State Key Laboratory of Computer Architecture, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing 100190)
(School of Computer Engineering, University of Chinese Academy of Sciences, Beijing 100190)

Abstract

Graph analytics power a range of applications in areas as diverse as machine learning, data mining and network security. However, due to the irregular memory access patterns and the large scale of graph data, memory accessing has become the bottleneck of graph applications. The vertex degree of graph follows a power-law distribution, and many recent researches use this property to store high degree vertexes in adjacent position of memory by graph reordering to increase spatial and temporal locality of memory access. However, graph reordering usually destroys the community structure in original graph, causing the performance degradation in structure graph datasets. To overcoming limitations of existing reordering techniques, the area graph reordering is proposed, which is a novel lightweight reordering technique that divides graph into areas before reordering to protect the original community structure of graph and increase the spatial and temporal locality. The proposed reordering algorithm is evaluated through testing six different real graph datasets and three graph applications on an Intel server. The results show that the proposed reordering technique yields average speed-up of 18.86% and 11.3%, compared with the baseline of no reordering and the state-of-the-art reordering technique based on vertex degree, seperately.

Key words: graph reordering, vertex degree, memory access locality, power-law distribution, community structure