计算机科学
COMPUTER SCIENCE

面向并发图分析的局部性感知的缓存管理策略

李汉桥, 赵苑君

引用本文

李汉桥, 赵苑君. 面向并发图分析的局部性感知的缓存管理策略[J]. 计算机科学, 2025, 52(12): 125-132.

LI Hanqiao, ZHAO Yuanjun. Locality-aware Cache Management Strategy for Concurrent Graph Analysis [J]. Computer Science, 2025, 52(12): 125-132.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250200062

面向并发图分析的局部性感知的缓存管理策略

李汉桥[1]　赵苑君[2]
1 武昌首义学院基础科学部　武汉 430064
2 华中科技大学　武汉 430074
(48250591@qq.com)

摘　要　随着图计算技术的蓬勃发展,现有图平台上通常运行着大量的并发图分析任务以获得数据背后的价值。因此,并发图计算技术被广泛应用于智能教育、公共管理和新闻媒体等领域。然而,目前图计算系统大多为执行单个图分析任务设计,在支持并发图分析任务时存在大量冗余数据访问。尽管一些工作已经观察到这一问题,并试图利用其中的时间和空间局部性共享底层图数据减少冗余数据访问,但是其忽视了私有状态数据更新访问的数据局部性,依然面临着缓存利用率低的问题,导致系统吞吐率低。为此,提出了面向并发图分析的局部性感知的缓存管理策略 CCG,以充分感知并发图分析任务之间的时间和空间局部性,减少冗余数据访问和同步开销。具体而言,该策略通过高效缓存数据的更新并以增量的方式合并冗余更新,利用并发图分析任务的数据局部性,实现内存数据的高效批量更新,减少数据访问开销并避免缓存抖动,有效提升并发图分析任务的吞吐率。同时,高效利用多级缓存进行分层缓冲与合并,让并发图分析任务在更新访问私有数据时避免同步开销和锁开销,进一步提升系统吞吐率。实验结果显示,在用目前最新的并发图计算系统 Glign 运行并发图分析任务时,相比于现有最好的图计算缓存策略 GRASP,CCG 可以将系统吞吐率提升 2.3～7.8 倍。

关键词:并发图分析;缓存架构;数据局部性;缓存抖动;吞吐率
中图分类号　TP391

Locality-aware Cache Management Strategy for Concurrent Graph Analysis

LI Hanqiao[1] and ZHAO Yuanjun[2]
1 Basic Science Department, Wuchang Shouyi University, Wuhan 430064, China
2 Huazhong University of Science and Technology, Wuhan 430074, China

Abstract　With the rapid growth of graph computing, modern graph platforms routinely execute a large number of concurrent graph analytics tasks to extract the latent value in massive datasets. Consequently, concurrent graph processing has been widely adopted in domains, including intelligent education, public administration, and news media. However, most existing graph processing systems are originally designed for single-task execution and suffer from excessive redundant data accesses when handling concurrent workloads. Although prior studies have observed significant redundancy in in-memory graph data across concurrent tasks and have attempted to exploit temporal and spatial locality to share underlying graph data, they largely overlook the data locality in private state updates. This limitation leads to low cache utilization and, ultimately, degraded system throughput. To address this challenge, this paper proposes CCG, a locality-aware cache management strategy for concurrent graph analysis, which fully exploits both temporal and spatial locality across tasks to reduce redundant data accesses and synchronization overhead. Specifically, CCG efficiently buffers and incrementally merges redundant updates, leveraging data locality to perform high-throughput batch updates in memory. This design minimizes access costs, mitigates cache thrashing, and significantly improves concurrency performance. Moreover, CCG employs a multi-level cache hierarchy to enable layered buffering and merging, thereby eliminating synchronization and locking overhead during private state updates. Experimental results show that CCG improves system throughput by 2.3×～7.8× over GRASP.

Keywords　Concurrent graph analysis, Cache architecture, Data locality, Cache thrashing, Throughput

1　引言

图算法通过对庞大的图数据集进行分析,能揭示关联数据背后的关键信息,被广泛应用于教育、管理、新闻等领域,如用于影响力评估的 PageRank 算法[1]、媒体推荐的 adsorption 算法[2]、数据聚类的连通分量算法[3]和社区并行发现算法[4]

到稿日期:2025-02-17　返修日期:2025-04-02
通信作者:赵苑君(138250205@qq.com)

等。然而,在真实场景中,大量的图算法可能被提交到同一图数据分析平台,以针对同一图数据进行不同方面的分析,为各种应用产品提供关键的信息服务。尽管众多高效的图计算系统[5-7]在过去十年中被开发出来,但目前主流的图计算系统通常按顺序处理各图分析任务,导致在处理并发图分析任务时无法感知到彼此,常常需要在缓存中重复加载相同的图结构数据,面临严重的数据访问瓶颈,并且未能充分利用底层硬件资源。这种不足导致了在执行并发图分析任务时,系统的数据访问开销巨大,缓存命中率低,支持并发图分析任务时吞吐率低。

一些并发图计算系统被提出,以消除内存中存在的大量冗余图数据。例如,Seraph[8]提出了一种图交换状态模型,并实现了图结构与图分析任务的解耦,从而提高了底层图数据的可重用性。CGraph[9],GraphM[10]和 Krill[11]提出子图级或数据块级等共享机制,进一步改善了内存冗余数据存储。Glign[12]通过对齐感知的批处理提高了图数据访问共享率,因此与其他系统相比可以充分共享图数据。这些系统尽管在支持并发图分析场景时有一定效果,但在支持并发图分析任务执行时,它们仅利用共享公共数据进行优化,并发图分析任务访问私有数据时仍然面临内存到最后一级缓存(Last Level Cache,LLC)的冗余数据访问和缓存抖动导致的缓存命中率低下等难题,系统吞吐量低。

与此同时,一些面向图计算的缓存管理策略[13-18]被提出。但是,这些缓存管理策略在支持并发图分析场景时,依旧面临着高额的随机访问开销和同步开销。首先,由于不同并发图分析任务极不规则的独立数据访问行为,内存数据相比于现有单个图分析任务场景更加随机且不规则,致使加载到 LLC 的数据被更加频繁地换进换出,尽管绝大多数加载到 LLC 中的数据还没有被使用。另外,并发图分析任务的激烈缓存争用行为导致同步开销高,系统吞吐量低。

针对这些问题,本研究提出了一种面向并发图分析的局部性感知的缓存管理策略 CCG(Locality-aware Cache Management Strategy for Concurrent Graph Analysis)。CCG 通过高效缓存数据的更新并以增量的方式合并冗余更新,批量执行内存数据更新,减少了并发图分析任务的随机访问和同步开销,有效降低了并发图分析任务在执行过程中的冗余数据访问。为了进一步减少数据访问次数,其提出多级缓存分层缓冲与合并策略,高效利用多级缓存结构进行分层缓冲与合并,让并发图分析任务在更新访问私有数据时避免同步开销和锁开销。实验结果显示,在用目前最新的并发图计算系统 Glign[12]运行并发图分析任务时,相比于现有最好的图数据缓存策略 GRASP[18],CCG 可以将并发图分析场景的系统吞吐率提升 2.3~7.8 倍。

本文的主要贡献如下:

1)研究了并发图分析中存在的高额随机访存开销和同步开销问题;

2)提出了一种面向并发图分析的局部性感知的缓存策略 CCG;

3)实现并评估了 CCG 原型,并通过实验证明了 CCG 的有效性。

2 相关工作

2.1 研究现状

2.1.1 并发图计算优化方法

真实应用中存在很多针对同一图数据进行并发迭代处理的复杂场景,这些并发图分析任务存在不规则的遍历顺序以及激烈的资源竞争的问题,导致底层图数据被各个任务反复加载至 LLC 中,激烈的资源争用使得底层内存子系统的利用率不足,最终导致整个系统的吞吐量很低。针对这些问题,现有方案使用软件或硬件方式,通过规则化并发图分析任务的数据访问行为,在并发图分析任务之间共享图数据,减少数据访问次数。

Seraph[8]提出将图结构与特定查询的数据解耦,以允许并发查询评估共享公共图结构数据。GraphM[10]提出了一种高效的共享-同步机制以及相应的预处理方法,将图结构数据划分为一系列逻辑块,让多个任务在 LLC 和内存中以相同的顺序访问图分区,从而减少了数据访问成本;同时,设计了一种调度策略,以便在多个任务之间充分利用数据访问的相似性,沿着相同的图路径进行遍历,从而提高系统的吞吐量。Krill[11]提出了一种分离模型,将图结构、算法描述和属性数据进行了分离。通过这种模型,程序员可专注于算法实现,方便地访问图结构和属性数据,而无需了解数据是如何存储的。Glign[12]通过对齐感知的批处理和重要迭代估计等技术,自动对齐并发图分析任务的不同图遍历,提升图的访问共享率,解决在相同类型并发图分析任务中底层图遍历可能错位的问题。

尽管上述并发图计算优化工作允许并发任务在每个迭代周期中共享对公共图数据的访问,可以减少数据重复加载至缓存的频率,但不同任务在各迭代中处理的数据往往有差异,不同图分析任务在每个迭代中所需处理的图数据通常是不同的;并且上述工作也没有考虑到在并发图分析任务更新自身私有状态数据时产生的随机访问,导致缓存频繁更新,相同的图数据仍会被重复加载到 LLC 进行处理。多核处理场景下更新私有数据,依旧面临着高额的同步开销问题。

2.1.2 面向图计算的缓存优化方法

为了优化图分析任务执行,研究者们已经提出了一些针对特定图算法的缓存性能提升策略。例如:Park 等[13]提出了针对 Floyd-Warshall,Bellman-Ford,Prim 和 Bipartite 匹配等算法的缓存优化方法;Safro 等[14]通过在并发广度优先搜索之间共享公共计算来提高缓存命中率,从而优化多源 BFS 算法。然而,这些方法主要针对特定算法,不适用于一般的图算法。

此外,一些研究致力于通过图排序和节点聚类来提升图分析任务执行的效率。Banerjee 等[15]提出了一种基于子深度优先遍历的节点排序方案,以提高有向无环图遍历的效率。P-OPT[16]提出了一种基于邻接矩阵的高性能图数据管理策略,利用图的邻接矩阵做出近似最优的缓存管理决策。DRRIP[17]通过预测缓存块的重引用间隔来优化缓存管理策略,解决了处理大数据集或频繁非临时数据引用的应用程序时的性能问题。GRASP[18]通过在现有缓存策略中增加修改

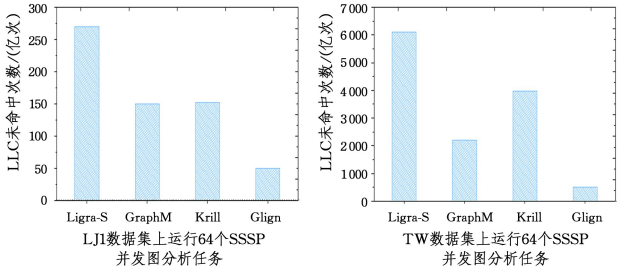
以优先处理包含高度顶点的缓存块，保护它们免受缓存抖动的影响，弥补了现有硬件缓存管理方案在处理具有不规则访问模式的图计算应用时的不足。

综上所述，现有图计算缓存优化策略仅针对单个图分析任务，没有考虑到并发图分析任务之间的关联性，无法充分利用并发图分析任务图数据之间的局部性，亦无法避免并发图分析任务之间的随机访问和缓存争用问题，导致大量冗余数据访问，缓存利用率低，并发图分析任务更新私有数据的同步开销大。

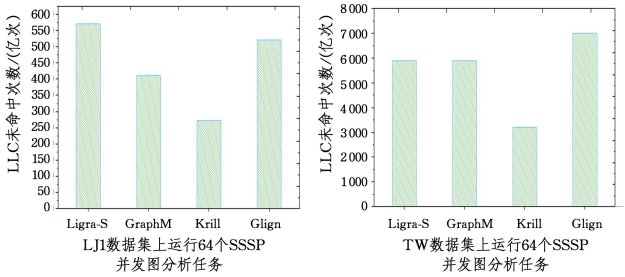
2.2 问题与挑战

并发图分析任务迭代过程需要访问公共数据和私有数据。公共数据指在多个并发图分析任务中共享的图结构数据，表示为 $G=(V,E,W)$，其中 V 表示图顶点 ID 数据，E 表示边数据，W 表示顶点和边权重数据。公共数据通常采用 CSR[19-20] 格式存储。私有数据指每个图分析任务专有的状态数据。例如：PageRank 算法中私有数据是排名值，连通分量算法中私有数据是联通分量 ID。

然而，使用现有方法时，并发图分析任务的非规则数据访问行为会导致随机访问。图 1 展示了运行在 Ligra-S[10]，GraphM[10]，Krill[11] 和 Glign[12] 上的 64 个并发图分析任务 LLC 未命中的情况。结果表明，虽然目前最好的并发图计算系统 Glign 相对于其他系统，在减少相同并发图分析任务缓存未命中率上性能优势较为明显，但其针对不同图分析任务并发执行时的性能差。这是因为现有方法往往仅针对并发图分析场景中公共数据的共享访问进行优化。然而，并发图分析任务在执行时更需要访问和更新大量私有数据。私有数据缓存未命中占所有缓存未命中的 78%～88%，导致并发图分析场景缓存命中率低。



(a)执行相同并发图分析任务集合的运行结果



(b)执行不同并发图分析任务集合的运行结果

图 1 64 个并发任务执行时 LLC 未命中的情况

Fig. 1 LLC misses during execution of 64 tasks

下面继续探究并发图分析任务集合加载到 LLC 中的私有状态数据缓存行被多个并发图分析任务所共享访问的比例。图 2 展示了并发图计算系统 Glign[12] 分别使用 LRU (Least Recently Used)，DRRIP[17] 和 GRASP[18] 缓存管理策略时的性能。可以看到，使用 GRASP 缓存管理策略的加速比最大。这是因为 GRASP 具有针对图结构特点的特定缓存优化。另外，测试结果显示加载到 LLC 中的私有数据缓存行中只有不到 0.001%的数据被复用超过两次。这意味着加载到缓存中的绝大多数私有数据在被换出之前未被使用，私有数据被频繁地反复加载进 LLC 中，产生了大量冗余访问。

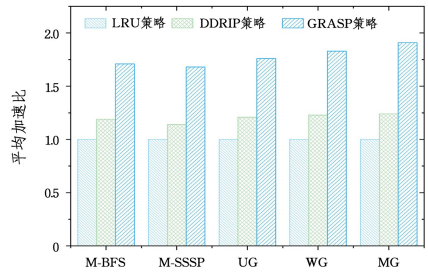


图 2 缓存管理策略对性能的影响

Fig. 2 Impact of cache replacement strategy on performance

由此可以看到，并发图分析任务执行时面临严重的数据访问问题。现有并发图计算优化机制都在试图通过共享公共数据的访问来减少重复加载到 LLC 中的数据量。然而，由于图算法种类繁多，不同图分析任务访问同一图顶点的时间不同，并发图分析任务执行时除了需要加载公共数据之外，还需要加载每个任务所需的私有顶点状态数据。在这一过程中，对于单个图分析任务而言，加载进 LLC 的数据是它需要的。然而，对于其他图分析任务来说，大部分加载进缓存的数据是无效数据，导致加载进 LLC 的私有数据缓存行利用率低。私有顶点状态数据被频繁加载进 LLC 中和更新到主存上，导致缓存抖动，产生严重的随机访问问题，带来高额的内存到 LLC 数据的传输量。

此外，并发图分析场景激烈的缓存争用，会导致同步开销高。在多核环境下，当一个核心完成对某个高度图顶点的计算后，该图顶点的状态数据会从 L2 缓存写回到 LLC 中。同时，其他核心也可能需要将相同顶点的状态数据写回 LLC，这时核心之间会出现激烈的资源竞争。在现有的缓存架构下，为了确保数据的一致性，当一个核心写回 LLC 时，必须进行加锁，这会带来巨大的锁开销和同步开销，降低多核环境下并发图分析任务的执行效率，并且浪费大量的计算资源。

3 面向并发图分析的局部性感知的缓存管理策略

为有效支持并发图分析任务执行，本文提出了面向并发图分析的局部性感知的缓存管理策略 CCG，包括面向并发图分析任务的批量更新策略和多级缓存分层缓冲与合并策略，以减少冗余数据访问和同步开销。

3.1 面向并发图分析任务的批量更新策略

为了解决并发图分析任务访问私有数据的随机性问题，CCG 提出了一种新颖的面向并发图分析的批量更新策略。该策略允许缓存扩展其功能来充当部分更新的合并写入缓冲区，以缓存图数据的增量更新，然后批量地更新内存中的私有数据。如算法 1 所示，当缓存收到针对非缓存行的更新请求时，它不会简单地获取整个缓存行，而是会缓冲部分更新，并

将它们合并到同一缓存行中(第2—5行),而只有在逐出时才将缓存更新行合并到内存中(第6—9行)。这种方法能够有效利用并发图分析任务中的时间局部性和空间局部性,从而显著减少批处理操作和向内存流式传输更新的次数。

算法1 面向并发图分析任务的批量更新策略

```
1. classcache:
2. def update_cache(addr,update_value):
3.     If addr not in self.cache:
4.         cache_store[addr]=0
5.     Acc(cache_store[addr],update_value)
6. def merge_to_memory(memory):
7.     foreach addr in self.cache:
8.         Acc(memory[addr],update_value)
9.     cache_store.clear()
```

具体而言,为了充分利用并发图分析任务的时间和空间局部性,其允许缓存缓冲区存储更新数据,而无需从内存中获取正在更新的数据。设计的缓存策略应当充当一个大型的合并写入缓冲区,如图3所示,$C_{t_i}^{\Delta}$表示t_i时刻后的高度顶点C的缓存更新行,C_{t_i}表示t_it_i时刻后C顶点的私有数据的状态值,图分析任务i中顶点C的状态数据表示为C_i。设计的缓存策略专门用于处理缓存未命中时的更新。传统的写入缓冲区通常包含全部数量的条目,并且无法有效捕获更新重用——这种重用往往在较长的时间范围内发生。因此,缓存策略旨在通过感知并发图分析任务的数据局部性,合并更新,优化缓存的使用,减少对内存的访问次数,从而提高系统的整体性能。

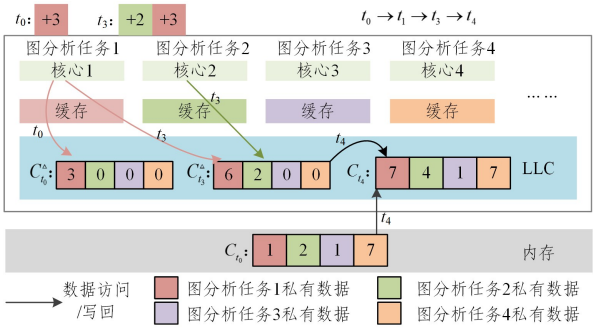


图3 面向并发图分析的批量更新策略

Fig. 3 Batch update strategy for concurrent graph processing analysis

当CPU核心在t_0时刻发现C_1缓存未命中时,它不必从内存中读取该地址的数据。相反,它可以直接发出更新指令,这样缓存就会生成一个包含批量更新的行地址为范围内所有并发图分析任务的C顶点状态数据$C_{t_i}^{\Delta}$。需要注意的是,此时缓存中的数据并不代表内存中相应地址的实际数据,而是表示一系列待处理的更新。随着时间的推移,这些更新将陆续到达,例如t_3时刻将C_1更新加3,将C_2更新加2,并将这些更新合并起来,使得缓存更新行$C_{t_3}^{\Delta}$数据变为(6,2,0,0)。当然,具有缓冲更新的缓存行不能像普通缓存行那样被替换或逐出,因为它们包含的是待处理的更新而非最终数据。在逐出该缓存行时,先从内存读取顶点C的私有状态向量(1,2,1,7),再与缓存中的更新向量(6,2,0,0)按元素相加,得到写回值(7,4,1,7)。

下面通过一个具体案例来探讨,这里并发图分析任务执行2个PageRank算法来计算网页的相对重要性。在此算法中,每个顶点在迭代过程中仅在其PageRank值发生显著变化时才会活跃。通过选择包含仅在上一次活跃时PageRank值发生显著变化的顶点的子集,可以加速计算。在现有方法中,对于PR-D 1和PR-D 2任务而言(以下简称PR-D 1和PR-D 2),假设PR-D 1和PR-D 2都会在不同迭代批次内反复更新高度顶点C,而当PR-D 2更新顶点C时距离PR-D 1更新距离较远。如果缓存在PR-D 2更新顶点C时顶点C的数据已被逐出,那么在PR-D 2更新C顶点时,会发生缓存未命中,导致必须重新从内存中读取顶点C。然而,根据本文提出的方法,对于高度连接的顶点,其更新将在缓存内进行缓冲。PR-D 1和PR-D 2对顶点C的更新将在缓存内合并,并且在合并后不会立即被逐出。相反,会在顶点C的大多数任务更新完成后,将所有更新合并再执行逐出策略,从内存中读取顶点C的顶点状态数据,并应用这些更新,将其更新到内存上去。其次,对于每个任务普通顶点的访问更新,在私有缓存中将更新收集起来进行批量更新,可以实现普通顶点私有数据的高局部性。因此,面向并发图分析的批量更新策略分为两个阶段。在第一阶段,每个核心将更新推送到LLC中,这里的LLC可以选择直接应用更新,或者将它们保留以进行批处理。在第二阶段,更新以批次的形式进行,每个批次对应于顶点在缓存中的部分。在这一阶段,核心会应用按批次的更新,以实现良好的局部性。这种策略不仅利用了并发图分析任务的时间局部性,而且能够有效地支持本文提出的其他优化措施,提高整体计算效率。

3.2 多级缓存分层缓冲与合并策略

为了解决并发图分析任务访问私有数据的缓存争用问题,以消除锁开销和同步开销,CCG提出并发图数据多级缓存分层缓冲与合并策略,利用私有和共享缓存在缓存层次结构的多个级别缓冲和合并更新,缓存级别之间无需同步,也无需更改一致性协议。如算法2所示,在进行更新时,每个核心首先将更新推送到私有缓存(第2—3行)。在此之后,私有缓存驱逐缓冲更新,并将这些更新合并到最后一级缓存(第4—7行)。最后一级缓存将对应的数据合并到主存(第8—9行)。

算法2 多级缓存分层缓冲与合并策略

```
1. class MultiCoreCacheSystem:
2. def update_private_cache(cid,addr,update_value):
3.     private_caches[cid].update_cache(addr,update_value)
4. def evict_to_llc(cid,addr):
5.     if addr inprivate_caches[cid].cache_store:
6.         update_value=private_caches[cid].cache_store.pop(addr)
7.         llc_cache.update_cache(addr,update_value)
8. def merge_llc_to_memory(memory):
9.         llc_cache.merge_to_memory(memory)
```

在多核处理器中的分层缓冲和合并操作如图4所示,每个内核都有自己的专用缓存。每个核心将更新推送到其私有缓存,它会分配一个缓冲更新行来保存更新。在图4中,$C_{t_i}^{\Delta}$表示t_i时刻后的顶点C的缓存更新行,C_{t_i}表示t_i时刻后C顶点的私有数据状态值,图分析任务i中顶点C的状态数据表示为C_i,$C_jt_i^{\Delta}$表示C_j在t_i时刻后的缓存更新行,在t_0时刻,图分

析任务 1，2，3 均更新了 C_1，C_2，C_3 的值，3 个私有缓存分别为 C_1，C_2，C_3 数据分配了一个缓冲更新行，而没有与 LLC 进行任何通信。然后，私有缓存可以将之后到来的其他更新合并到同一行，正如前文所述的方法一样，当在 t_1 时刻，图分析任务 1 更新了 C_1 的值时，可以直接在私有缓存级别进行更新合并。当在 t_2 时刻，私有缓存需要替换或驱逐某个缓冲更新行时，它只需将这些更新作为更新消息发送到 LLC。LLC 随后会根据情况分配一个新的缓冲更新行，或者将接收到的更新与现有缓冲行中的更新合并。这样的合并可以同时发生，其中 $C^{\Delta}_{t_2}$ 包含顶点 C 全部私有数据的更新。

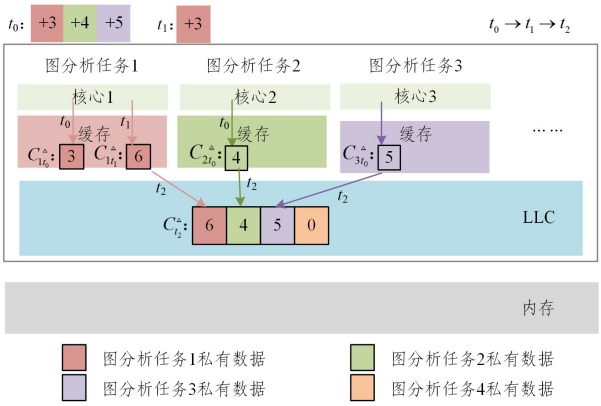
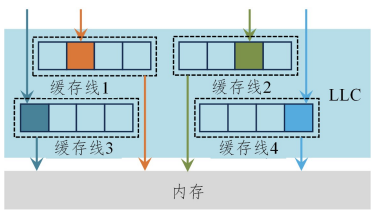


图 4 多级缓存分层缓冲与合并策略

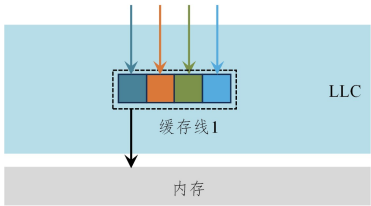
Fig. 4 Multi-level cache buffering and merging strategy

由于并发图分析任务的空间局部性，这种在私有缓存和 LLC 多级缓存中进行分层缓冲与合并的策略，可以有效缓解由于不规则图顶点状态数据访问导致的资源竞争，同时可以有效减少高度图顶点状态数据的更新次数。

如图 5 所示，在现有方法中，缓存数据更新较为碎片化，需要多次写入操作来满足不同任务的私有数据更新。然而，文中的方法能够将多个任务的私有数据合并到一个缓存线中一次性写入，从而提高了数据更新时的局部性。



(a)现有方法的缓存数据更新



(b)批量缓存数据合并更新

图 5 批量缓存数据合并更新

Fig. 5 Merging of batched cache updates

除此之外，所提出的方法还简化了多核系统中的并发数据处理，从而显著提升了并发图分析任务的执行性能，避免了并发图分析任务之间的缓存争用导致的同步开销。

最后，量化了所提出的方法对缓存利用率的提升：

$$U=\frac{\sum_{i\in I}\sum_{C\in V}U_C^i}{\sum_{i\in I}\sum_{C\in V}T_C^i} \tag{1}$$

其中，U 为有效缓存命中率，I 表示所有并发图任务的集合，V 表示所有顶点的集合，U_C^i 代表任务 i 中顶点 C 在缓存中的有效命中更新次数，T_C^i 代表任务 i 中顶点 C 的总更新次数。

相比现有方法，文中所提方法能够有效提升 U_C^i，从而提升缓存命中率。

4 性能评估

为了验证 CCG 的性能，使用 ZSim[21] 进行微架构模拟，并使用表 1 中列出的参数模拟一个 64 核系统。处理器采用乱序处理器核心，使用模仿英特尔 Skylake 处理器核心建模并验证。每个核心都有专用的 L1 和 L2 缓存，所有核心共享一个 64 MB 的 LLC，处理器核心扩展支持 AVX512 指令，然后在模拟的处理器上实现了 CCG，Glign 由 g++ 5.3 进行编译，同时启用-O3。

表 1 系统具体配置

Table 1 System configuration

组件	配置
CPU 核心	64 核，X86-64 ISA，2.2 GHz，Skylake-like OOO
L1 缓存	每个核 32 KB，8 路组相连，时延：3 个时钟周期
L2 缓存	每个核私有 256 KB，12 路组相连，时延：6 个时钟周期
LLC	所有核心共享 64 MB，16 路哈希组相连，时延：27 个时钟周期
内存	12 通道 DDR4 3200 CL17

如表 2 所列，实验选取 4 个真实图数据集[10]，分别用 WT，LJ1，TW 和 UK 表示。实验使用并发图分析任务使用频率较高的 6 个图算法作为测试基准[10]，分别为 SSSP，CC，BFS，DFS，PR 和 BC。这些算法既包含从部分顶点开始激活遍历的图算法，也包含从全部顶点开始遍历的图算法，同时包含带权图和无权图的遍历算法。这些算法对内存带宽消耗也不同，例如 BFS，DFS 和 CC 内存带宽消耗少，而 SSSP，BC 和 PR 内存带宽消耗多；同时，这些算法访问顶点次序不同，迭代收敛速度也不同。实验分别测试同时提交相同图算法和不同图算法的并发图分析任务。并对于相同的 BFS 和 SSSP，选择不同的源顶点。

表 2 数据集描述

Table 2 Description of datasets

数据集	顶点数量	边数量
WT	2 394 385	5 021 410
LJ1	4 847 571	68 993 773
TW	41 652 230	1 468 365 182
UK	105 896 555	3 738 733 648

将上述 6 个算法组合成 5 组不同的并发图分析任务集合，每组集合包含 12 个并发图分析任务，以尽可能地覆盖真实世界可能会出现的场景。如表 3 所列，第一组为 12 个 BFS 任务，表示为 M-BFS；第二组为 12 个 SSSP 任务，表示为 M-SSSP；第三组为 4 个 BFS 任务、4 个 DFS 任务和 4 个 CC 任务，因为这些任务对内存带宽需求少，且均可以运行在无权图上，所以将其表示为无权图组(UG)；第四组为 4 个 SSSP 任务、4 个 PR 任务和 4 个 BC 任务，因为这些任务对内存带宽

需求多，均需要运行在有权图上，所以将其表示为带权图组(WG)；最后一组为 BFS 任务、DFS 任务、CC 任务、SSSP 任务、PR 任务、BC 任务各 2 个，表示为混合类型任务组(MG)，并发图分析任务使用动态提交方式，提交时间节点服从泊松分布。

表 3 并发图分析任务集合

Table 3 Concurrent graph analysis task sets

并发图分析任务集合	并发图分析任务描述
M-BFS	{BFS}×12
M-SSSP	{SSSP}×12
UG	{BFS,DFS,CC}×4
WG	{SSSP,PR,BC}×4
MG	{BFS,DFS,CC,SSSP,PR,BC}×2

首先，在 ZSim 模拟硬件环境下运行最新的并发图计算系统 Glign[12]，其采用 GRASP[18] 缓存管理策略而不包含 CCG 扩展，分别对 4 个数据集和 5 个并发图分析任务集合进行了测试。然后，在 CCG-G(即运行最新的并发图计算系统 Glign[12]，并且此时采用 CCG 扩展)上也进行相应的对照实验。

从图 6 可见，相比于采用 GRASP 缓存管理策略的 Glign，CCG-G 可以获得 2.3～7.8 倍的性能提升。在混合任务提交时，CCG-G 优势更大。也就是说，现有方法只在并发图分析任务全为 SSSP 或 BFS 等情形时可以提升任务的数据访问利用率，性能提升较大。但是，当任务差异性较大时，数据访问行为无法被现有方法有效规则化，而 CCG 主要在缓存架构层面进行了优化，使用的缓存内合并和批量更新技术对差异性较大的任务一样有效，改善了不同类型图分析任务执行时的空间局部性，可以提升数据利用率，有效减少数据访问次数。

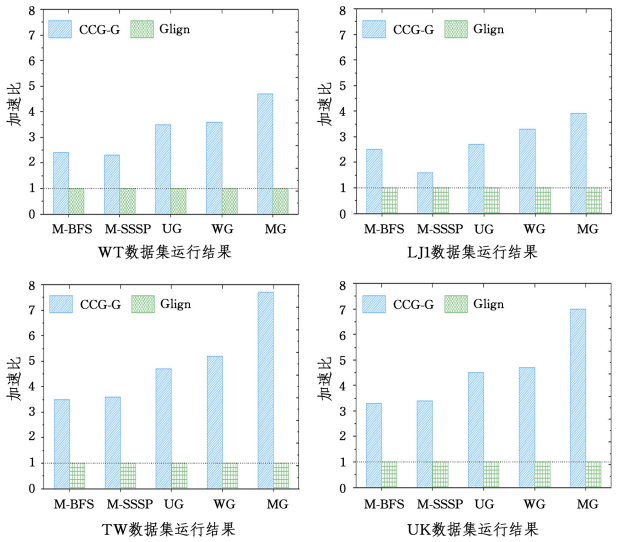


图 6 在 4 个不同数据集上运行不同并发图分析任务集合时 CCG-G 和 Glign 的性能对比

Fig. 6 Performance of CCG-G and Glign running different concurrent graph analysis task sets on four different datasets

实验显示，Glign 的内存访问次数是 CCG-G 的 6.62 倍。这是因为 Glign 仅在读的过程中针对公共数据部分进行共享，然而更新过程中并发图分析任务需要更新自己的私有数据，导致对一个图顶点的更新频繁进行，缓存在写和读之间频繁切换。这种现象在多核资源争用中更为突出。相比而言，CCG 能减少数据访问次数，并在多核之间通过更新合并避免高额同步开销，提升计算资源利用率。

同时，从图 6 中可以发现，CCG-G 相比于 Glign 的性能加速比在 TW 数据集上最高。这是因为 TW 数据集顶点平均度数更高，数据倾斜度更高。由于并发图分析任务之间的局部性，对高度顶点的更新会更加频繁，也更适合 CCG 在分层缓存结构进行更新合并，减少冗余数据访问。图 7 和图 8 分别展示了 CCG-G 和 Glign 在 TW 数据集上的 LLC 未命中率和内存访问次数。

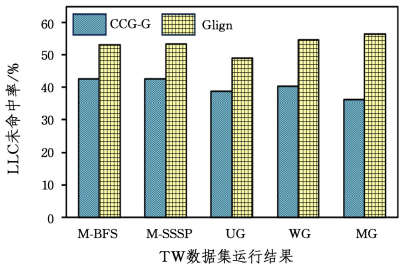


图 7 CCG-G 和 Glign 在 TW 数据集上的 LLC 未命中率

Fig. 7 LLC miss rate of CCG-G and Glign on TW

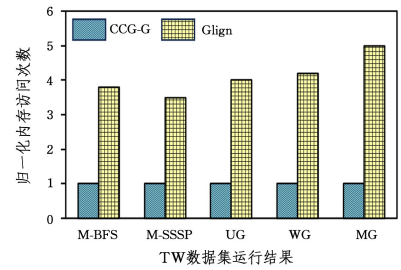


图 8 CCG-G 和 Glign 在 TW 数据集上的内存访问次数

Fig. 8 Memory accesses of CCG-G and Glign on TW

最后，对影响 CCG 性能的各种因素的敏感性进行分析。图 9 展示了不同缓存大小对 CCG 性能的影响。结果显示，在缓存较小时，CCG 的提升较大，这是因为 CCG 大幅度提升了缓存空间的利用率，但是当缓存增大后，CCG 性能上升较缓。图 10 展示了高度顶点阈值 α(即度数前百分之几的顶点算作高度顶点)对 CCG 性能的影响。从图 10 可见，CCG 的加速效果会先增加后减少。当 α 值太小时，被合并存储的高度顶点变少，无法充分利用高度顶点状态合并表，带来了更多的额外数据访问开销。相反，当 α 值太大时，被合并存储的高度顶点变多，导致 LLC 中的数据利用率降低，浪费了 LLC 空间。对于不同数据集而言，由于倾斜度不同，因此最佳的 α 值也不同。

图 11 展示了不同数量 PR 任务和不同数量 BFS 任务下，CCG-G 在 TW 数据集上相对于 Glign 的加速比。可见，并发任务数增加后，CCG-G 相对于 Glign 可以获得更好的加速比。这主要是由于 Glign 的缓存命中率随着并发任务数量增加急剧下降，然而 CCG-G 仍然可以很好地保证高命中率。这说明 CCG 更适合并发图分析任务数量多的情况。此外，从图 11 可见，CCG-G 对于计算密集型任务(如 PR)的加速效果不及

内存密集型任务（如 BFS），说明 CCG 更适合内存密集型任务。

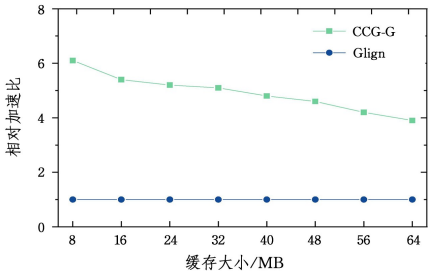


图 9 缓存大小对 CCG-G 和 Glign 性能的影响

Fig. 9 Impact of cache size on CCG-G and Glign

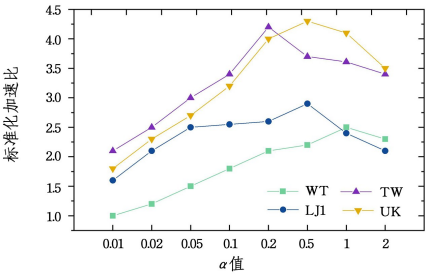


图 10 高度顶点阈值对 CCG 性能的影响

Fig. 10 Impact of α on CCG

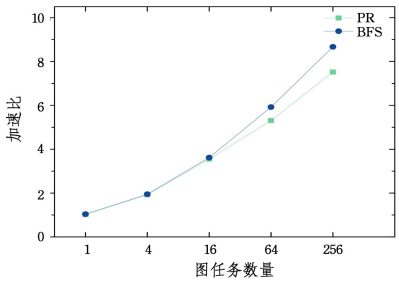


图 11 不同数量图任务时 CCG-G 在 TW 上相对于 Glign 的加速比

Fig. 11 Speedup ratio of CCG-G compared with Glign on TW under different number of graph tasks

结束语 本文提出了面向并发图分析的局部性感知的缓存管理策略，以充分感知并发图分析任务之间的时间和空间局部性，减少并发图分析任务执行过程中的冗余数据访问和同步开销。实验结果显示，相对于现有最好的方法，所提方法能使得并发图分析任务吞吐率得到进一步提升。未来拟进一步对其进行扩展，使其支持动态图场景。

参 考 文 献

[1] PAGE L, BRIN S, MOTWANI R, et al. The PageRank citation ranking: bringing order to the web[R]. Stanford: Stanford InfoLab, 1998.

[2] ZHANG Y, GAO Q, GAO L, et al. Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 25(8): 2091-2100.

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

[4] QIAO S J, GUO J, HAN N, et al. Parallel Algorithm for Discovering Communities in Large-Scale Complex Netwoks[J]. Chinese Journal of Computers, 2017, 40(3): 687-700.

[5] YANG K, ZHANG M X, CHEN K, et al. Knightking: a fast distributed graph random walk engine[C]// Proceedings of the 2019 ACM Symposium on Operating Systems Principles. 2019: 524-537.

[6] ZHANG Y, LIAO X, GU L, et al. AsynGraph: Maximizing data parallelism for efficient iterative graph processing on GPUs[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2020, 17(4): 1-21.

[7] LOW Y, GONZALEZ J, KYROLA A, et al. Distributed graphlab: A framework for machine learning in the cloud[J]. arXiv: 1204.6078, 2012.

[8] XUE J, YANG Z, QU Z, et al. Seraph: an efficient, low-cost system for concurrent graph processing[C]// Proceedings of the 2014 International Symposium on High-performance Parallel and Distributed Computing. 2014: 227-238.

[9] ZHANG Y, LIAO X, JIN H, et al. {CGraph}: A correlations-aware approach for efficient concurrent iterative graph processing [C]// Proceedings of the 2018 USENIX Annual Technical Conference. 2018: 441-452.

[10] ZHAO J, ZHANG Y, LIAO X, et al. GraphM: an efficient storage system for high throughput of concurrent graph processing [C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019: 1-14.

[11] CHEN H, SHEN M, XIAO N, et al. Krill: a compiler and runtime system for concurrent graph processing[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021: 1-16.

[12] YIN X, ZHAO Z, GUPTA R. Glign: Taming misaligned graph traversals in concurrent graph processing[C]// Proceedings of the 2022 ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 78-92.

[13] PARK J S, PENNER M, PRASANNA V K. Optimizing graph algorithms for improved cache performance[J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(9): 769-782.

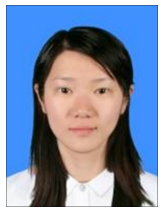
[14] SAFRO I, TEMKIN B. Multiscale approach for the network compression-friendly ordering [J]. Journal of Discrete Algorithms, 2011, 9(2): 190-202.

[15] 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.

[16] BALAJI V, CRAGO N, JALEEL A, et al. P-opt: Practical optimal cache replacement for graph analytics[C]// Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture. IEEE, 2021: 668-681.

[17] JALEEL A, THEOBALD K B, STEELY JR S C, et al. High performance cache replacement using re-reference interval prediction (RRIP)[J]. ACM SIGARCH Computer Architecture News, 2010, 38(3): 60-71.
[18] FALDU P, DIAMOND J, GROT B. Domain-specialized cache management for graph analytics[C]// Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture. IEEE, 2020: 234-248.
[19] HAM T J, WU L, SUNDARAM N, et al. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics[C]// Proceedings of the 2016 Annual IEEE/ACM International Symposium on Microarchitecture. IEEE, 2016: 1-13.
[20] NGUYEN D, LENHARTH A, PINGALI K. A lightweight infrastructure for graph analytics[C]// Proceedings of the 2013 ACM Symposium on Operating Systems Principles. 2013: 456-471.
[21] SANCHEZ D, KOZYRAKIS C. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems[J]. ACM SIGARCH Computer architecture news, 2013, 41(3): 475-486.

LI Hanqiao, born in 1983, master, associate professor. His main research interests include federated learning, graph neural networks and augmented reality.



ZHAO Yuanjun, born in 1988, master. Her main research interest is software system.

(责任编辑:柯颖)

CCF 数字图书馆编审委员会 2025 年工作会议在哈尔滨召开

2025 年 10 月 24 日,CCF 数字图书馆编审委员会 2025 年工作会议在哈尔滨华旗饭店顺利举行。

CCF 会士、数图编委主任苏金树,CCF 副秘书长王新霞和各专委秘书长、常务委员及代表齐聚,围绕数图运营优化、资源建设与未来发展展开研讨,锚定“强化知识传播、构建高质量数字资源生态”核心目标。会议由 CCF 助理秘书长兼出版部主任韩飞主持。

王新霞副秘书长强调,数图建设依托 CCF 核心志愿者团队,学会将持续保障技术与资源投入,“以编委认可的内容与服务,打造行业优质平台”。

数图编委主任苏金树提出“让数图成为中国计算机研究者必备工具”,指出需突破会员权限、宣传广度等瓶颈,凸显数图知识传播使命。

韩飞主任明确“3M 原则”,以“资源收录+宣传推广”为双核心,推动数图从“资源积累”向“服务升级”跨越。同时,会上发布了年度核心成果:

资源规模跨越式增长:收录 CCF 会刊过刊论文 17 万篇(较去年增 14 万篇),视频资源近 1 万个(年度新增超 1000 个),CNCC 大会等核心活动实现“当日报告次日看回放”。

功能迭代升级:新增智能推荐、免费科普专区、视频智能转写等功能,解决内容检索痛点。

传播成效显著:4—9 月 PV 月均增长超 30%,焦点专题带动总流量破 30 万,实现专业内容大众化传播。

会议期间,苏金树主任主持数图编委表彰环节,对过去一年工作表现突出的 11 位编委予以表彰,肯定他们在资源对接、宣传推广等数图建设关键环节的贡献,进一步激发全体编委的参与热情。

与会者围绕数图高质量发展深入研讨,明确核心方向 ——以提升内容质量与传播广度为关键,提出设立专题报告、加强与专委合作、推动视频精细化加工、拓展学生群体影响力等措施,进一步增强数图价值。同时针对激励机制、宣传推广、技术趋势分析、高校图书馆合作、国际化标签等深度议题展开交流,构建完善的数字资源生态体系。

据 CCF 微信公众号