POSTER: Domain-Specialized Cache Management for Graph Analytics

Priyank Faldu
The University of Edinburgh
priyank.faldu@ed.ac.uk

Jeff Diamond
Oracle Labs
jeff.diamond@oracle.com

Boris Grot
The University of Edinburgh
boris.grot@ed.ac.uk

***Abstract*—In the domain of graph analytics, power-law graphs are prevalent. In such graphs, a small fraction of vertices are responsible for a large share of all graph connections. These richly-connected *(hot)* vertices inherently exhibit high reuse. However, this work finds that the state-of-the-art hardware cache management schemes struggle in capitalizing on their reuse due to highly irregular access patterns of graph analytics. In response, we argue in favor of leveraging software knowledge of graph data structures to accurately pinpoint hot vertices in hardware. To that end, we propose GRASP, a domain-specialized LLC management scheme that enables high cache efficiency for graph analytics with minimal modifications to existing cache structures.**

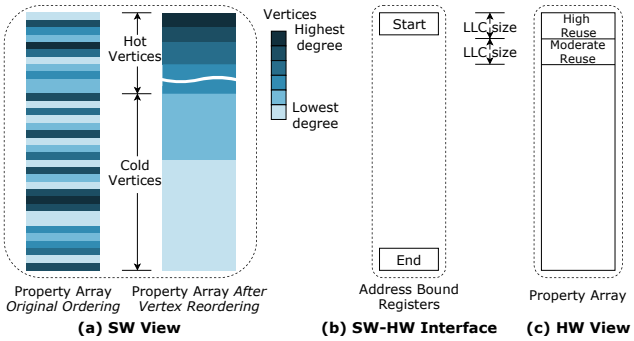

Figure 1: GRASP overview.

I. Introduction

A distinguishing property of graph datasets common in many graph-analytic applications is that the vertex degrees follow a skewed power-law distribution, in which a small fraction of vertices (*hot vertices*) have many connections while the majority of vertices (*cold vertices*) have relatively few connections. Such graphs are prevalent in a variety of domains including social networks and computer networks.

Graph applications are notorious for exhibiting irregular access patterns. When processing large graphs, accesses to a large number of cold vertices cause severe *cache thrashing*, often forcing hot vertices out of the cache. Thus, protecting hot vertices against cache thrashing is crucial for a hardware cache management scheme to improve cache efficiency, and in turn, application performance.

The state-of-the-art hardware schemes employ domain-agnostic prediction-based mechanisms [5, 8, 3] to identify high-reuse cache blocks. However, we find that graph-dependent irregular access patterns prevent these schemes from correctly learning which cache blocks to preserve, rendering them deficient for graph analytics. Meanwhile, a recent work [4] proposes *pinning* high-reuse cache blocks in LLC to protect them from thrashing. However, we observe that pinning-based schemes are overly rigid when applied to graph processing and result in sub-optimal cache utilization.

To address these limitations, we propose GRASP – Graph Specialized LLC management. GRASP augments existing cache policies to provide preferential treatment to cache blocks containing hot vertices to shield them from thrashing. To cater to the irregular access patterns, GRASP policies are inherently flexible and allow for caching of other blocks exhibiting reuse. To accurately pinpoint hot vertices in hardware, GRASP leverages existing software vertex reordering techniques [1, 2, 6], and exposes a lightweight software-hardware interface. GRASP does not require any additional metadata at the LLC or storage-intensive prediction tables, making GRASP attractive for commercial adoption.

II. GRASP : Caching in on the Skew

GRASP observes that existing software skew-aware vertex reordering techniques [1, 2, 6] induce spatial locality among hot vertices by segregating them in a contiguous memory region as shown in Fig. 1(a). GRASP leverages the contiguity in designing a lightweight interface, which aids hardware in pinpointing hot vertices. Meanwhile, GRASP augments insertion and hit-promotion policies of a baseline cache management scheme (e.g., RRIP [9]) to protect hot vertices from thrashing. Overall, GRASP design consists of three hardware components as follows.

A. Software-Hardware Interface

Prior works characterize cache access patterns for various graph data structures and show that LLC and main memory requests are dominated by accesses to the *Property Array(s)* that holds the partial or fully computed results for all vertices, making Property Array(s) the prime target for caching [1, 2]. GRASP exposes a pair of *Address Bound Registers (ABR)* per Property Array, which are part of an application context. To enable GRASP, the graph framework populates each ABR pair with the start and end virtual address of the entire Property Array (Fig. 1(b)). Once ABRs are populated, GRASP does not require any further software intervention.

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00051

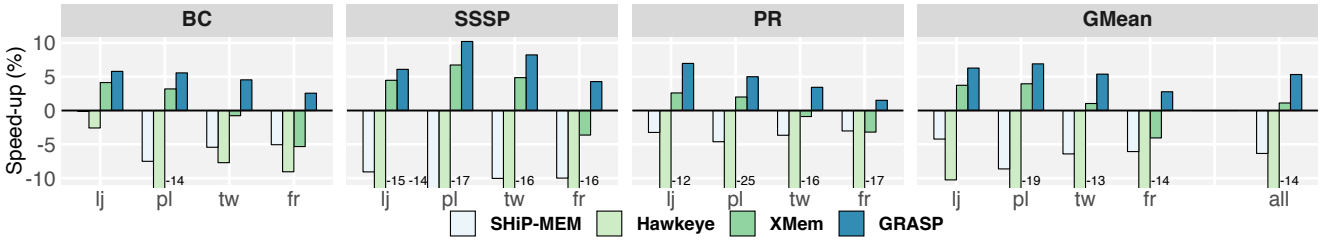

Figure 2: Speed-up for GRASP and the state-of-the-art cache management schemes over the RRIP [9] baseline.

B. Classification Logic

Pinpointing the High Reuse Region: GRASP labels two LLC-sized sub-regions within the Property Array. A region at the start of the Property Array is labeled as *High Reuse Region*; another one starting immediately after the High Reuse Region is labeled as *Moderate Reuse Region* (Fig. 1(c)). Finally, if an application specifies more than one Property Array, GRASP divides LLC-size by the number of Property Arrays before labeling the regions.

Classifying LLC Accesses: At runtime, GRASP classifies a memory address making an LLC access as *High-Reuse* if the address belongs to the High Reuse Region of any Property Array; GRASP determines this by comparing the address with the bounds of the High Reuse Region of each Property Array. Similarly, an address is classified as *Moderate-Reuse* if the address belongs to the Moderate Reuse Region. All other LLC accesses are classified as *Low-Reuse*. When ABRs are not set, all accesses are classified as *Default*. GRASP encodes the classification result as a 2-bit *Reuse Hint*, and forwards it to the LLC along with each cache request.

C. Specialized Cache Policies

Insertion Policy: LLC miss tagged as High-Reuse is inserted in the LLC at the MRU position to protect it from thrashing. Meanwhile, LLC miss tagged as Moderate-Reuse is inserted near the LRU position while LLC miss tagged as Low-Reuse is inserted at the LRU position.

Hit-Promotion Policy: LLC hit tagged as High-Reuse is immediately promoted to the MRU position whereas any other LLC hit only gradually promotes associated cache block towards MRU position with every hit.

Eviction Policy: GRASP does not modify eviction policy of the baseline scheme, which avoids the need to explicitly store the Reuse Hint as additional LLC metadata while also keeping the cache management flexible.

III. Evaluation

We evaluate three graph applications – Betweenness Centrality (BC), Single Source Shortest Path (SSSP) and Pagerank (PR) – from the Ligra framework [7]. Each application processes four datasets having 5M–64M vertices and 68M–2147M edges, resulting in 12 datapoints. We reorder datasets using DBG [1] and simulate one iteration with the highest number of active vertices for each application-dataset pair. We evaluate GRASP and three state-of-the-art thrash-resistant schemes – SHiP-MEM [8], Hawkeye [3], and XMem [4] – and report the speed-up over RRIP [9] using the Sniper simulator modeling 8 OoO cores with 16-way 16MB NUCA (2MB slice per core).

As Fig. 2 shows, GRASP yields an average speed-up of 5.3% (up to 10.2%) over the RRIP baseline. Over the same baseline, SHiP-MEM, Hawkeye and XMem yield an average speed-up of -6.4%, -14.1% and 1.1%, respectively. GRASP outperforms all the evaluated techniques across the datapoints with an average speed-up of 4.2% over XMem, 5.3% over RRIP, 12.5% over SHiP-MEM and 22.7% over Hawkeye.

Prior predictive schemes cause slowdown on all datapoints with max slowdown of 13.6% for SHiP-MEM and 24.6% for Hawkeye over the RRIP baseline. The result indicates that the learning mechanisms of the domain-agnostic schemes are deficient in identifying, and in turn, retaining the high reuse working set (i.e., hot vertices) for graph applications. Meanwhile, among prior schemes, XMem outperforms Hawkeye and SHiP-MEM, which confirms our thesis that utilizing software knowledge for cache management is a promising direction over storage-intensive domain-agnostic design for the challenging access patterns of graph analytics.

References

[1] P. Faldu et al. “A Closer Look at Lightweight Graph Reordering”. In: *IISWC*. 2019.
[2] V. Balaji et al. “When is Graph Reordering an Optimization? Studying the Effect of Lightweight Graph Reordering Across Applications and Input Graphs”. In: *IISWC*. 2018.
[3] A. Jain et al. “Rethinking Belady’s Algorithm to Accommodate Prefetching”. In: *ISCA*. 2018.
[4] N. Vijaykumar et al. “A Case for Richer Cross-Layer Abstractions: Bridging the Semantic Gap with Expressive Memory”. In: *ISCA*. 2018.
[5] P. Faldu et al. “Leeway: Addressing Variability in Dead-Block Prediction for Last-Level Caches”. In: *PACT*. 2017.
[6] Y. Zhang et al. “Making Caches Work for Graph Analytics”. In: *Big Data*. 2017.
[7] J. Shun et al. “Ligra: A Lightweight Graph Processing Framework for Shared Memory”. In: *PPoPP*. 2013.
[8] C.-J. Wu et al. “SHiP: Signature-based Hit Predictor for High Performance Caching”. In: *MICRO*. 2011.
[9] A. Jaleel et al. “High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP)”. In: *ISCA*. 2010.

This work is partially supported by a grant from Oracle Labs.