Brief Announcement: Block-Granularity-Aware Caching

Nathan Beckmann
beckmann@cs.cmu.edu
Carnegie Mellon University

Phillip B. Gibbons
gibbons@cs.cmu.edu
Carnegie Mellon University

Charles McGuffey
cmcguffe@cs.cmu.edu
Carnegie Mellon University

ACM Reference Format:
Nathan Beckmann, Phillip B. Gibbons, and Charles McGuffey. 2021. Brief Announcement: Block-Granularity-Aware Caching. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3409964.3461818

1 INTRODUCTION

A common feature of computer systems is that *block granularity* changes at different levels of the storage hierarchy. This paper presents the first study of how granularity change affects caching. We define the *Block-Granularity-Aware (BGA) Caching Model*; prove new adversarial competitive bounds for the problem; and develop an online BGA caching policy with a better competitive ratio than traditional cache policies in this setting.

Why does block granularity change? Given that a large and fast memory does not exist, real systems make use of a hierarchy ranging from small, fast memories to larger and slower storage devices [14]. Each level of the storage hierarchy organizes its data in blocks to simplify management and reduce overheads. For example, SRAM caches typically consist of 64 B "lines"; DRAM of 2-4 KB "rows"; and flash/disk of 4 KB "pages".[1]

Most caches today ignore granularity change and only load data of their own granularity. But this misses an opportunity to load some or all of the larger-granularity block at minimal cost (see Figure 1), as the lower level has already fetched the entire block. Motivated by this observation, we ask: *What caching opportunities and challenges are introduced by granularity change?*

What does prior caching work say about block granularity? The original caching problem is well studied and understood. Belady [4] and Mattson [19] separately devised optimal solutions for caching with unit-size and -cost items. Sleator and Tarjan [27] provided both lower and upper bounds for the cost ratio when comparing the performance of *online* caches, which must make decisions as requests arrive, against *offline* caches, which are allowed to view the entire trace when making decisions. Fiat et al. [13] extended this work to randomized algorithms and showed ways of approximating online policies using other online policies.

[1]In fact, there can be different granularities for reads and writes, e.g., "erase blocks" in flash can be many MBs. We focus on reads in this work.

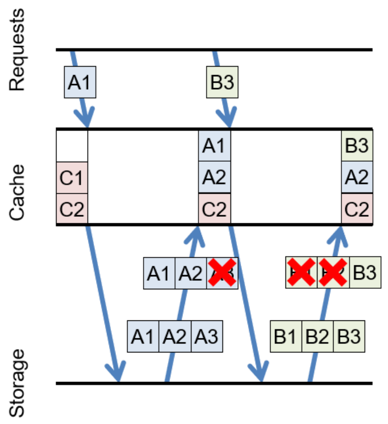

Figure 1: In the BGA Caching Problem, caches can choose what subset of the larger-granularity block from the level below them to load.

Other variants of caching have been considered, and considerable work has been done on complexity and algorithms for these variants [1, 3, 6, 9, 11, 32]. Caching with variable-size items [5, 10, 33] appears to be similar to BGA Caching, since one could think of different subsets of a block as differently sized items. The critical difference is that, unlike in variable-size caching, items in a block can be accessed, cached, and evicted individually. In BGA Caching, choosing which items to load is an additional dimension with significant impact on performance. To our knowledge, there is no prior theoretical work that accounts for granularity change.

The systems community uses several approaches to handle granularity change. There is work on scheduling memory controllers at granularity boundaries [12, 22, 23, 35–37], address mapping techniques [18, 29, 31, 34], row-buffer management [21, 24, 28, 30], and item-to-block allocation [2, 7, 8, 25]. The most relevant of these works are DRAM caches that account for granularity change by taking some or all of the larger-granularity block into the smaller-granularity cache on loads [16, 17, 26]. We provide the first theoretical framework to better understand and guide these designs.

Contributions. In this work, we investigate the effects of granularity change on caching. Our results include the following:

- We develop a model for caching at a granularity boundary, called the Block-Granularity-Aware (BGA) Caching Problem.
- We show that the Offline BGA Caching Problem is NP-Complete.
- We provide a lower bound on the competitive ratio of deterministic replacement policies in BGA Caching.
- We design and analyze Item-Block Layered Partitioning, a practical BGA caching policy, and we prove an upper bound that is much tighter than any policy that considers only a single granularity.

For a more in-depth discussion of our results, see [20].

2 THE MODEL

The Block-Granularity-Aware Caching Model consists of a single level of memory (cache) that receives a series of requests, referred to as accesses, to data items. If the item is in the cache, then the request is served and the cache is not charged. If the item is not in the cache, then the cache must load the item from the subsequent level of memory or storage. If this load causes the amount of data in cache to exceed the cache size k, then items must be evicted from the cache to remedy the situation.

What makes the Block-Granularity-Aware Caching Model unique is that the universe of items is partitioned into *blocks* of up to B items. When the cache loads data from storage, it can load any subset of the block for unit cost; i.e., items after the first are "free". When each item is in a different block, this model exactly matches the traditional caching model.

The blocks represent the larger data granularity used by the subsequent level of the memory hierarchy. In such systems, there is typically a small memory buffer used to handle data as it is being read or written. The cost of moving data from bulk storage into this buffer is typically large relative to the cost of operating on the buffer itself. Hence, once items are brought into the buffer, they can be accessed at low cost, motivating our model [14, 15].

Definition 2.1. In the *Block-Granularity-Aware Caching Problem*, we are given *(i)* a cache of size k, *(ii)* an (online or offline) trace σ of requests to items, and *(iii)* a partitioning of the items into disjoint blocks (sets) such that no partition contains more than B items. Starting with an empty cache, the goal is to minimize the number of times that an item is not present in the cache when requested in σ. When a requested item is not in cache, any subset of that item's block can be loaded, so long as the subset contains the item.

Locality vs. traditional caching models. In traditional caching models, all hits come from *temporal locality*, i.e., when an item remains in cache between subsequent accesses. In the BGA Caching Model, hits can also come from *spatial locality*, i.e., when an item is in cache due to an earlier access to a different item in the same block. (Any hits beyond the first are due to temporal locality, since the item would have been brought in cache anyway.)

Baseline policies. We consider two baseline cache designs. An *Item Cache* loads only the requested item from a block; i.e., it is a traditional cache. By contrast, a *Block Cache* loads all the items in a requested block and also evicts them together; i.e., it increases the cache's granularity to operate on blocks instead of items. Item Caches perform well on temporal locality and poorly on spatial locality, whereas Block Caches are the opposite.

3 COMPLEXITY ANALYSIS

We show that the Block-Granularity-Aware Caching Problem is NP-Complete using a reduction from variable-size caching.

Theorem 3.1. *The Offline BGA Caching Problem is NP-Complete.*

We are able to simulate variable-size items using multiple items from the same block accessed consecutively. By repeating these access sequences, we force the optimal solution to load all used items in the block. The optimal solution to the generated instance can easily be translated into an optimal solution to the input instance.

Point	Sleator-Tarjan Bound	BGA Lower Bound	BGA Upper Bound
Constant Augmentation	$k = 2h \Rightarrow 2\times$	$k \approx 2h \Rightarrow B\times$	$k \approx 2h \Rightarrow 2B\times$
Ratio = Augmentation	$k = 2h \Rightarrow 2\times$	$k \approx \sqrt{B}h \Rightarrow \sqrt{B}\times$	$k \approx \sqrt{2B}h \Rightarrow \sqrt{2B}\times$
Constant Ratio	$k = 2h \Rightarrow 2\times$	$k \approx Bh \Rightarrow 2\times$	$k \approx Bh \Rightarrow 3\times$

Table 1: Interesting bounds for online cache size k and optimal cache size h, shown as: Augmentation $\Rightarrow$ Competitive Ratio. Compared to traditional caching, the spatial locality in BGA Caching adds a penalty of $\Theta(B)\times$ to the product of competitive ratio $\times$ augmentation.

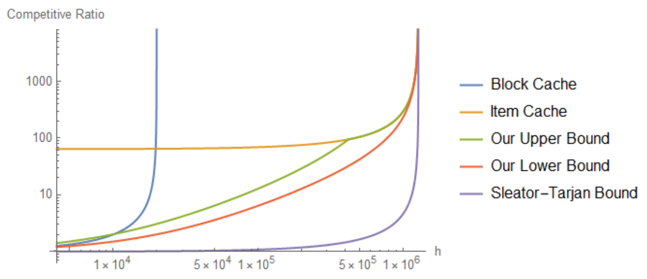

Figure 2: Comparing bounds in the BGA Caching Problem. The x-axis is the optimal cache size h, and the y-axis is the competitive ratio. Online cache size $k = 1.28M$ and block size $B = 64$.

4 COMPETITIVE LOWER BOUND

Theorem 4.1. *The competitive ratio of any deterministic policy is at least* $(k + (B - 1)(h - 1))/(k - h + 1)$ *where* k *is the size of the online cache and* $h \le k - B + 1$ *is the size of the optimal cache.*

For large caches with large blocks ($k > h \gg B \gg 1$), the lower bound is roughly $(k+Bh)/(k-h)$. As seen in Figure 2, this bound is much greater than the Sleator-Tarjan [27] bound, meaning that the gap between online and offline policies is larger in BGA Caching than in traditional caching. The difference from the traditional bound is the $Bh/(k-h)$ term. The gap starts at a multiplicative factor of nearly $B\times$ when $k \approx h$ (since the Bh term dominates), and tapers off, hitting $2\times$ when $k \approx Bh$. Table 1 gives some interesting points of comparison for the Sleator-Tarjan bound, our lower bound, and our upper bound: constant factor augmentation, the point where the augmentation meets the competitive ratio, and constant competitive ratio. These results show that, compared to traditional caching, the introduction of spatial locality increases the gap between online and offline policies by $B\times$, which can be spread between the competitive ratio and the augmentation factor.

An additional insight that we gain from our lower bound (see [20] for more details) is that in order to maximize performance, policies should load the entire block to take advantage of spatial locality, but evict items individually to take advantage of temporal locality by favoring those that have been accessed over those that have not.

5 COMPETITIVE UPPER BOUND

5.1 Policy Description

Our policy, Item-Block Layered Partitioning (IBLP), divides the available space into two different layers of cache. The first layer serves each access to the cache by loading only the *items* that are accessed. The second layer only serves accesses that miss in the first layer, but loads and evicts at the granularity of *entire blocks* at a time. We refer to these layers as the item layer and block layer, respectively, and define their sizes as i and b. Both layers perform evictions using the Least-Recently Used (LRU) replacement policy.

5.2 The Upper Bound

Theorem 5.1. *The competitive ratio of IBLP is upper bounded by:*

$$\begin{cases} \frac{(b+B(2i-1))^2}{8B(B+b)(i-h)} & i \leq \frac{2Bb-b+2B^2+B}{2B} \\ \frac{2Bi-Bb+b-B^2-B}{2i-2h} & i > \frac{2Bb-b+2B^2+B}{2B} \end{cases}$$

where $i \geq h$ is the size of the item layer, b is the size of the block layer, and h is the size of the optimal cache.

We use this bound to partition the cache space between layers.

Known optimal size. When the size of the optimal cache is known, the optimal layer sizes can be directly computed. When $k \geq \frac{3Bh-h-B^2-B}{B-1}$, this results in:

$$\text{Ratio} = \frac{(k+B-1)(k-h+B(2h-1))}{(k-h+B)^2}$$

$$i = \frac{k^2+4Bhk-hk+4B^2h-3Bh-B^2}{2Bk+k+2Bh-h+2B^2-3B}$$

For smaller k values, setting $i = k$ (i.e., operating as an Item Cache) provides the minimum competitive ratio of:

$$\frac{2Bk-B^2-B}{2(k-h)}$$

This transition occurs at the point where the competitive ratio due to temporal locality exceeds the maximum competitive ratio that can be achieved due to spatial locality ($B\times$). In other words, for small k values (relative to h), temporal locality dominates performance.

Again considering large caches with large blocks ($k > h \gg B \gg 1$), we see that the ratio is roughly $\frac{k(k+2Bh)}{(k-h)^2}$ if $k \geq 3h$ and $\frac{Bk}{k-h}$ if $k < 3h$.

Figure 2 shows how this upper bound compares to the lower bound, as well as single-granularity caches of the same size running LRU. IBLP outperforms the small-granularity Item Cache for $k \approx 3h$ and larger, and it outperforms the large-granularity Block Cache for $k \approx 4Bh$ and smaller. In addition, IBLP performs close to optimal for all values of k, whereas the performance of the baselines degrades severely outside of their ideal performance conditions.

Table 1 shows how this bound compares to traditional caching and our lower bound. Our upper bound has roughly the same penalty to augmentation and competitive ratio as our lower bound, differing by at most a multiplicative factor of 3×.

Unknown optimal size. The optimal layer sizes in IBLP depend on the size of the optimal cache h being compared against. For any fixed layer sizes, the competitive ratio will be optimal at only one value of h, but show significant degradation for larger h and limited improvement for smaller h.

This dependency is unique amongst known caching problems. Unlike other caching problems, the competitive ratios due to temporal locality and spatial locality are different functions of the optimal cache size. As a result, the relative performance of traces changes depending on optimal cache size, which results in different values of h having different worst-case traces. This suggests that a full understanding of the BGA Caching Problem requires analysis beyond competitive ratios.

ACKNOWLEDGMENTS

Supported in part by NSF grants CCF-1919223, CCF-2028949, a Google Research Scholar Award, a VMware University Research Fund Award, and by the Parallel Data Lab (PDL) Consortium (Alibaba, Amazon, Datrium, Facebook, Google, Hewlett-Packard Enterprise, Hitachi, IBM, Intel, Microsoft, NetApp, Oracle, Salesforce, Samsung, Seagate, and TwoSigma).

REFERENCES

[1] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. 1999. Page replacement for general caching problems. In *SODA*.
[2] Hagit Attiya and Gili Yavneh. 2017. Remote memory references at block granularity. In *OPODIS*.
[3] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. 2001. A unified approach to approximating resource allocation and scheduling. *JACM* (2001).
[4] Laszlo A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal* (1966).
[5] Daniel S. Berger, Nathan Beckmann, and Mor Harchol-Balter. 2018. Practical Bounds on Optimal Caching with Variable Object Sizes. In *SIGMETRICS*.
[6] Mark Brehob, Stephen Wagner, Eric Torng, and Richard Enbody. 2004. Optimal replacement is NP-hard for nonstandard caches. *IEEE Trans. Comput.* (2004).
[7] Brad Calder, Chandra Krintz, Simmi John, and Todd Austin. 1998. Cache-conscious data placement. In *ASPLOS*.
[8] Trishul M Chilimbi, Mark D Hill, and James R Larus. 1999. Cache-conscious structure layout. In *PLDI*.
[9] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. 1991. New Results on Server Problems. *SIAM Journal on Discrete Mathematics* (1991).
[10] Marek Chrobak, Gerhard J. Woeginger, Kazuhisa Makino, and Haifeng Xu. 2012. Caching is hard-even in the fault model. *Algorithmica* (2012).
[11] Guy Even, Moti Medina, and Dror Rawitz. 2018. Online generalized caching with varying weights and costs. In *SPAA*.
[12] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. 2001. Memory controller policies for DRAM power management. In *ISLPED*.
[13] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A McGeoch, Daniel D. Sleator, and Neal E. Young. 1991. Competitive paging algorithms. *Journal of Algorithms* (1991).
[14] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach.*
[15] Bruce Jacob, David Wang, and Spencer Ng. 2010. *Memory systems: cache, DRAM, disk.*
[16] Djordje Jevdjic, Gabriel H Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison cache: A scalable and effective die-stacked DRAM cache. In *MICRO*.
[17] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. 2013. Die-stacked DRAM caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache. In *ISCA*.
[18] Wei-Fen Lin, Steven K Reinhardt, and Doug Burger. 2001. Designing a modern memory hierarchy with hardware prefetching. *IEEE Trans. Comput.* (2001).
[19] Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. 1970. Evaluation techniques for storage hierarchies. *IBM Systems journal* (1970).
[20] Charles McGuffey. 2021. Modernizing Models and Management of the Memory Hierarchy for Non-Volatile Memory. (2021).
[21] Seiji Miura, Kazushige Ayukawa, and Takao Watanabe. 2001. A dynamic-SDRAM-mode-control scheme for low-power systems with a 32-bit RISC CPU. In *ISLPED*.
[22] Onur Mutlu and Thomas Moscibroda. 2007. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*.
[23] Onur Mutlu and Thomas Moscibroda. 2008. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*.
[24] Seong-Il Park and In-Cheol Park. 2003. History-based memory mode prediction for improving memory performance. In *ISCAS*.
[25] Erez Petrank and Dror Rawitz. 2002. The hardness of cache conscious data placement. *POPL*.
[26] Moinuddin K Qureshi and Gabe H Loh. 2012. Fundamental latency trade-off in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design. In *MICRO*.
[27] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* (1985).
[28] Vladimir V Stankovic and Nebojsa Z Milenkovic. 2005. DRAM controller with a close-page predictor. In *EUROCON*.
[29] Kshitij Sudan, Niladrish Chatterjee, David Nellans, Manu Awasthi, Rajeev Balasubramonian, and Al Davis. 2010. Micro-pages: increasing DRAM efficiency with locality-aware data placement. In *ASPLOS*.
[30] Ying Xu, Aabhas S Agarwal, and Brian T Davis. 2009. Prediction in dynamic SDRAM controller policies. In *SAMOS*.
[31] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A Harding, and Onur Mutlu. 2012. Row buffer locality aware caching policies for hybrid memories. In *ICCD*.
[32] Neal Young. 1994. The k-server dual and loose competitiveness for paging. *Algorithmica* (1994).
[33] Neal E. Young. 2002. On-line file caching. *Algorithmica* (2002).
[34] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. 2000. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *MICRO*.
[35] Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu. 2008. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *MICRO*.
[36] Zhichun Zhu and Zhao Zhang. 2005. A performance comparison of DRAM memory system optimizations for SMT processors. In *HPCA*.
[37] Zhichun Zhu, Zhao Zhang, and Xiaodong Zhang. 2002. Fine-grain priority scheduling on multi-channel memory systems. In *HPCA*.