Brief Announcement: Spatial Locality and Granularity Change in Caching

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

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

Charles McGuffey
cmcguffey@reed.edu
Reed College

CCS CONCEPTS

• **Computer systems organization** → *Architectures*; • **Theory of computation** → *Computability*; **Caching and paging algorithms**;

KEYWORDS

Caching; Spatial locality; Block granularity; Online algorithm

ACM Reference Format:
Nathan Beckmann, Phillip B. Gibbons, and Charles McGuffey. 2022. Brief Announcement: Spatial Locality and Granularity Change in Caching. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '22), July 11–14, 2021, Philadelphia, PA, USA.* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Real systems make use of a hierarchy ranging from small, fast memories to larger and slower storage devices [15]. Each level of the hierarchy organizes its data in blocks to simplify management and reduce overheads. The size of a block (*block granularity*) depends on the level. For example, SRAM caches typically consist of 64 B "lines", DRAM of 2-4 KB "rows", and flash/disk of 4 KB "pages". This brief announcement highlights our work [8, 19] providing the first theoretical study of how this *granularity change* affects caching.

What caching opportunities and challenges are introduced by granularity change? While caching policies traditionally ignore granularity change (in both theory and practice), there is an emerging trend in systems to optimize for it [5, 17, 18, 20–22, 24–26]. Granularity change provides an optimization 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 [17, 18, 22]. But the heuristics used by these systems are best-effort and have no theoretical bounds. We provide the first theoretical framework to better understand and guide these designs.

What does prior caching work say about block granularity? The original caching problem is well studied and understood. Sleator and Tarjan [23] 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. Work on both complexity and algorithms has also been done on several caching variants [2, 6, 10–12, 27].

To our knowledge, no prior theoretical work accounts for granularity change. Prior work focuses on the temporal locality of items within the access trace and misses the significant impact of spatial locality among data items in the level below. Models of computation that account for transfers to and from the cache in "blocks" [1, 4, 9, 14] permit items in a block to be individually accessed, but not individually cached or evicted. As such, the "blocks" are the (smaller) granularity of the cache itself and not the (larger) granularity of the level below. Choosing which items to load is an additional dimension with significant impact on performance.

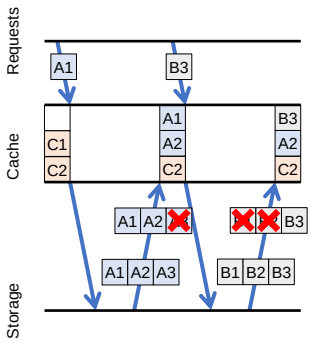

Figure 1: In Granularity-Change Caching, caches can load any subset of the larger-granularity block from the level below them, for the same cost.

Our results. We study the effects of granularity change on caching:

(i) We develop a model for caching at a granularity boundary, called the Granularity-Change (GC) Caching Problem;
(ii) We analyze competitive ratios in GC Caching, providing complexity results, a lower bound for determinisitic online policies, and an upper bound policy that outperforms any policy considering only a single granularity; and
(iii) Because these bounds reveal a new issue that arises in GC Caching where the choice of offline cache size affects the competitiveness of different online policies relative to one another, we develop a locality model for GC Caching that admits upper and lower bounds for miss rate based solely on the system's cache size and the workload.

This brief announcement focuses on contribution *(iii)*. A prior brief announcement [7] focused on contribution *(ii)*, and [8] gives a comprehensive treatment.

2 THE MODEL

The **Granularity-Change 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 (a *hit*), then the request is served and the cache is not charged. If the item is not in the cache (a *miss* or *fault*), then the cache must load the item from the subsequent level of memory or storage. If this load causes

SPAA '22, July 11–14, 2021, Philadelphia, PA, USA

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 Granularity-Change Caching Model unique is that the universe of items is partitioned into *blocks* of up to B items, such that the cache can load *any subset* of the items in a block for unit cost; i.e., items after the first are "free" (B=3 in Figure 1). When B=1, the 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 [15, 16].

Definition 2.1. In the **Granularity-Change 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 misses resulting from the requests 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 GC Caching, hits can also come from *spatial locality*, i.e., when an item I is in cache due to an earlier access to a *different* item in the same block.

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 COMPETITIVE RATIOS

We provide a theoretical framework for analyzing competitive ratios in the GC Caching Problem. For details see [7, 8].

Our first result is the complexity class of the offline problem:

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

We also show a general lower bound for deterministic policies:

Theorem 3.2. *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* $h \leq k - B + 1$ *is the size of the optimal cache.*

Finally, we provide an online policy, called Item-Block Layered Partitioning (IBLP), that divides the available space into two different layers of cache. The *item layer* serves each access to the cache by loading only the *items* that are accessed. The *block layer* only serves accesses that miss in the item layer, but loads and evicts at the granularity of *entire blocks* at a time. Both layers perform evictions using LRU.

Table 1: Salient bounds for online cache size k and optimal cache size h, shown as: Augmentation ▸ Competitive Ratio.

Setting	S-T Bound	GC LowerB	GC UpperB
Const. Augmentation	$k = 2h \triangleright 2\times$	$k \approx 2h \triangleright B\times$	$k \approx 2h \triangleright 2B\times$
Ratio=Augmentation	$k = 2h \triangleright 2\times$	$k \approx \sqrt{B}h \triangleright \sqrt{B}\times$	$k \approx \sqrt{2B}h \triangleright \sqrt{2B}\times$
Constant Ratio	$k = 2h \triangleright 2\times$	$k \approx Bh \triangleright 2\times$	$k \approx Bh \triangleright 3\times$

Theorem 3.3. *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.*

Table 1 gives three salient 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. We also note that the gap between our upper and lower bounds is at most a multiplicative factor of 3×.

A new problem with competitive ratios. Our analysis reveals a new issue that arises in GC Caching: the relative performance of online policies (i.e., different item/block partition sizes) changes with the size of the optimal cache they are compared against. This is problematic in practice—we would like to be able to design and analyze caches independent of a hypothetical comparator. To overcome this, we next consider an analysis based on the amount of temporal and spatial locality in the trace.

4 BEYOND COMPETITIVE RATIOS

We extend the locality of reference model by Albers et al. [3] to account for block granularity. Their model adds a function $f(n)$ that characterizes the number of distinct items accessed in a window of n accesses across a trace. They use this model to provide bounds on the miss rate (the number of misses per access) of various replacement policies as a function of $f(n)$. We extend this model by adding a similar function $g(n)$ to account for the number of distinct *blocks* accessed in a window of size n.

Theorem 4.1 (Lower Bound). *Any deterministic replacement policy has a miss rate of at least*

$$\frac{g(f^{-1}(k+1)-2)}{f^{-1}(k+1)-2}$$

Proof. We construct a family of traces matching the bound. Each trace uses $k+1$ distinct items. Due to the locality constraints, these items can be partitioned into at most $g(f^{-1}(k+1)-2)$ blocks. We generate traces in phases, where each phase consists of $f^{-1}(k+1)-2$ accesses divided into $k-1$ repetitions. A repetition consists of repeated accesses to a single item that has not yet been accessed this phase. In each phase, repetition $1 \leq j \leq k-1$ starts with the $f^{-1}(j+1)-1$th access of that phase and continues until the access before the next block starts. These traces are consistent with $f(n)$.

Table 2: Salient bounds for comparing an equally split cache ($i = b$) to the lower bound for a cache of half the size ($h = i + b$).

$f(n)$	$g(n)$	Lower Bound	Item-Layer UB	Block-Layer UB
$n^{1/2}$	$n^{1/2}$	$1/h$	$1/i$	B/b
$n^{1/2}$	$n^{1/2}/B^{1/2}$	$1/(B^{1/2}h)$	$1/i$	$1/b$
$n^{1/2}$	$n^{1/2}/B$	$1/Bh$	$1/i$	$1/Bb$
$n^{1/p}$	$n^{1/p}$	$1/h^{p-1}$	$1/i^{p-1}$	B^{p-1}/b^{p-1}
$n^{1/p}$	$n^{1/p}/B^{1/2}$	$1/(B^{(p-1)/p}h^{p-1})$	$1/i^{p-1}$	$1/b^{p-1}$
$n^{1/p}$	$n^{1/p}/B$	$1/Bh^{p-1}$	$1/i^{p-1}$	$1/Bb^{p-1}$

It remains to discuss $g(n)$ and show the minimum miss rate of policies on these traces. In the work of Albers et al. [3], the item is chosen such that it is not in the cache; thus every repetition causes one miss. In GC Caching, our ability to choose items is limited by the $g(n)$ function. In particular, the item not in cache may be from a block that has not been accessed yet in the phase. If this is the case, then choosing that item increases the number of blocks accessed in the window, which may cause a violation. However, it is known that a new block can be chosen at least $g(p)$ times for a phase of length p. Each of these new block choices returns us the freedom to guarantee that we can pick the item that is not in cache. □

We now provide an upper bound for the miss rate of IBLP. Because this policy consists of two layers of cache acting in concert, both layers must miss in order for the entire cache to miss. We therefore begin by providing bounds for each layer individually.

Lemma 4.2 (Item-Layer Upper Bound). *The miss rate of the item layer of size i is at most*

$$\frac{i-1}{f^{-1}(i+1)-2}$$

Proof. The item layer is simply a traditional LRU cache. The change in model cannot cause the miss rate to increase, because it only introduces new ways for a policy to hit. This means that we can rely on the result from Albers et al. [3] on bounds on LRU policies in the traditional model. □

Lemma 4.3 (Block-Layer Upper Bound). *The miss rate of the block layer of size b is at most*

$$\frac{b/B-1}{g^{-1}(b/B+1)-2}$$

Proof. The behavior of the block layer depends on the block that is accessed, but it is independent of the particular item. We can view the block layer as an LRU cache with effective size b/B, serving a trace where requests are to blocks instead of items. Simply substitute the effective cache size and $g(n)$ in Lemma 4.2. □

Taking the minimum of these miss rates for a given input provides an upper bound on the miss rate of IBLP.

Theorem 4.4 (Upper Bound). *The miss rate of IBLP with item layer size i and block layer size b is upper bounded by:*

$$\min\left(\frac{i-1}{f^{-1}(i+1)-2}, \frac{b/B-1}{g^{-1}(b/B+1)-2}\right)$$

As an example analysis, we consider how IBLP performs on polynomial locality functions (i.e., $f^{-1}(n) = cn^p$ for some real numbers c and p). Because locality functions must be positive concave functions, this covers the majority of high order terms that would occur in real traces. We choose the partitioning and augmentation parameter factors such that $i = b = h = k/2$. Table 2 shows the results of the analysis. At the extremes, where $f(n) = g(n)$ or $f(n) = Bg(n)$, one of the partitions matches the lower bound. The largest gap between the baseline and the upper bound for IBLP occurs when the ratio between $f(n)$ and $g(n)$ is $B^{1-(1/p)}$. As the value of p approaches ∞, this gap approaches B.

In addition to reinforcing our results from competitive ratios, this shows that the relative performance of IBLP is worst when locality is high. But with high locality, miss ratios are low. Therefore, the large relative gap results in a small number of additional misses. This suggests that IBLP will perform well in practice.

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.

REFERENCES

[1] Alok Aggarwal and Jeffrey S. Vitter. 1988. The Input/Output Complexity of Sorting and Related Problems. *Commun. ACM* 31, 9 (1988).
[2] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. 1999. Page replacement for general caching problems. In *SODA*.
[3] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. 2005. On paging with locality of reference. *J. Comput. System Sci.* 70, 2 (2005).
[4] Bowen Alpern, Larry Carter, and Jeanne Ferrante. 1993. Modeling Parallel Computers as Memory Hierarchies. In *Programming Models for Massively Parallel Computers*.
[5] Hagit Attiya and Gili Yavneh. 2017. Remote memory references at block granularity. In *OPODIS*.
[6] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. 2001. A unified approach to approximating resource allocation and scheduling. *JACM* 48, 5 (2001).
[7] Nathan Beckmann, Phillip B. Gibbons, and Charles McGuffey. 2021. Brief Announcement: Block-Granularity-Aware Caching. In *SPAA*.
[8] Nathan Beckmann, Phillip B. Gibbons, and Charles McGuffey. 2022. Spatial Locality and Granularity Change in Caching. *arXiv preprint arXiv:2205.14543* (2022).
[9] Guy E. Blelloch, Rezaul Alam Chowdhury, Phillip B. Gibbons, Vijaya Ramachandran, Shimin Chen, and Michael Kozuch. 2008. Provably good multicore cache performance for divide-and-conquer algorithms. In *SODA*.
[10] Mark Brehob, Stephen Wagner, Eric Torng, and Richard Enbody. 2004. Optimal replacement is NP-hard for nonstandard caches. *IEEE Trans. Comput.* 53, 1 (2004).
[11] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. 1991. New Results on Server Problems. *SIAM Journal on Discrete Mathematics* 4, 2 (1991).
[12] Guy Even, Moti Medina, and Dror Rawitz. 2018. Online generalized caching with varying weights and costs. In *SPAA*.
[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* 12, 4 (1991).
[14] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 1999. Cache-Oblivious Algorithms. In *FOCS*.
[15] John L. Hennessy and David A. Patterson. 2012. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann.
[16] Bruce L. Jacob, Spencer W. Ng, and David T. Wang. 2008. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann.
[17] Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison cache: A scalable and effective die-stacked DRAM cache. In *MICRO*.
[18] 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*.
[19] Charles McGuffey. 2021. *Modernizing Models and Management of the Memory Hierarchy for Non-Volatile Memory*. Ph.D. Dissertation. Carnegie Mellon University.
[20] Onur Mutlu and Thomas Moscibroda. 2007. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*.
[21] Onur Mutlu and Thomas Moscibroda. 2008. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*.
[22] 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*.
[23] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985).
[24] 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*.
[25] Ying Xu, Aabhas S. Agarwal, and Brian T. Davis. 2009. Prediction in dynamic SDRAM controller policies. In *SAMOS*.
[26] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A. Harding, and Onur Mutlu. 2012. Row buffer locality aware caching policies for hybrid memories. In *ICCD*.
[27] Neal Young. 1994. The k-server dual and loose competitiveness for paging. *Algorithmica* 11, 6 (1994).