Remote Memory Prefetching: Is Coarse-grained Fine?

James McMahon
u0975161@umail.utah.edu
University of Utah
Salt Lake City, Utah, USA

Vinita Pawar
vinita@cs.utah.edu
University of Utah
Salt Lake City, Utah, USA

Ryan Stutsman
stutsman@cs.utah.edu
University of Utah
Salt Lake City, Utah, USA

Abstract

CXL raises new questions about tiering, pooling, and remote memory access. In most disaggregated memory approaches, compute nodes access remote memory pools at page (4 KB) granularity. This is the case for two reasons: to help amortize high remote access costs and because the host CPUs' address translation hardware is set up for 4 KB pages. While fetching and caching whole remote pages helps with applications that have high spatial locality, for some applications it can introduce contention for cache capacity since potentially cold or unrelated adjacent data is also cached. Additionally, this can increase network bandwidth utilization.

Operating at a smaller cache block granularity (e.g. 64 B) could reduce remote access amplification and make more efficient use of local caches and the network. Further, operating at a cacheline granularity would allow future work to build upon the many decades of work done in CPU hardware data prefetching, since prefetchers at that level primarily operate at small cacheline granularities. Because of the higher latencies associated with remote memory access, more time and hardware resources can be used to generate prefetch predictions. This could allow previous models that were too complex for real-world CPU data prefetching to find practical application and for practical models to be scaled up.

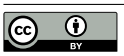
In this work, we explore whether cacheline-granular prefetching could be beneficial for memory pooling and far-memory systems. Specifically, we investigate whether it is possible to achieve performance comparable to today's page-granular remote accesses by using small, cacheline-granular remote accesses with aggressive prefetching. This paper shows that while cacheline-granular prefetching seems like a natural next step for remote CXL devices, beating page granular accesses appears to be difficult for the workloads we explored.

CCS Concepts

• **Hardware** → **Emerging interfaces**; *External storage*; *Networking hardware*; • **Computing methodologies** → *Distributed computing methodologies.*

Keywords

Compute Express Link (CXL); Prefetching; Storage; Memory Pooling; Remote Memory



ICPE Companion '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1130-5/2025/05
https://doi.org/10.1145/3680256.3721318

ACM Reference Format:
James McMahon, Vinita Pawar, and Ryan Stutsman. 2025. Remote Memory Prefetching: Is Coarse-grained Fine?. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '25), May 5–9, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3680256.3721318

1 Introduction

Compute Express Link (CXL) is a new standard that extends the long-standing PCI Express (PCIe) I/O interconnect with a new cache coherent protocol [5]. CXL has been the topic of many recent papers since it offers new ways to operate on data shared with CPU cores compared to PCIe's classic DMA and interrupt-based approach [10, 12, 25, 26].

In addition to allowing for coherent memory accesses between the host CPU and CXL devices, CXL also includes protocols for many hosts to share the memory capacity of many remote memory devices. As a result, works like Pond have explored using CXL for memory pooling [12].

With CXL supporting cacheline-granular accesses for far memory, it raises the question of how this would perform compared to classic approaches that access remote memory at page granularity. To that end, this paper uses simulation on common microbenchmark workloads to compare the performance of a classic demand-fetched, page-based approach to remote memory with one that uses cacheline-granular accesses combined with aggressive prefetching strategies. We move cacheline-granular prefetching past the last-level cache (LLC) to explore how capable existing prefetchers can be when their received data is filtered by the LLC. This, however, comes at the cost of disabling hardware prefetching to have a level playing field to make clear comparisons. Our results show that with large caches and high network bandwidth, accessing at page granularity performs well compared to a granular, prefetching-intensive approach, though in some cases the granular approach can match or beat the simple page-based solution.

2 Background

Sharing memory over a network is not a new idea [14]; early distributed shared memory systems shared memory over the network in order to solve problems that exceeded the resource capacities of a single machine. However, one key issue with sharing memory over a network is high access latencies. Remote memory accesses are much slower than local DRAM accesses, and this has limited the applications that can take advantage of remote memory. Latency-sensitive applications will prefer local memory to avoid suffering large performance hits. However, with CXL gaining traction, there has been renewed interest in far memory and memory pooling; this allows load and store access to memory on other machines connected via a network, potentially without software involvement.

More specifically, a compute node could have non-local memory, which it is operating on as if it was local memory.

One goal of using remote memory is to reduce costs by driving up CPU and memory utilization in clusters of data center machines. The under-utilization of hardware is a problem in large-scale computing because the hardware costs money, and not using the hardware to its full potential is a waste of the extra money spent [6, 12, 15]. Also, hardware energy consumption is often non-linear with utilization, meaning it is more power efficient to fully utilize the hardware [15]. Hardware under-utilization can come from the imbalance of compute needs and memory needs on a node. If the compute is overwhelmed, no more tasks can be scheduled, and so any remaining memory goes unused [12]; similarly, if the memory is overwhelmed, the remaining compute goes unused. In both cases, the hardware costs and the energy costs are being paid for hardware that isn't being used.

These are some of the cases that motivate CXL memory sharing. If a distributed system can share memory across multiple machines, the memory can be more efficiently utilized. A machine can be allowed to use the excess memory of another machine; similarly, a machine with too little memory can borrow memory from a memory pool [6, 12]. Ideally, this would solve the scheduling issue of trying to fill both CPU and memory usage for each individual machine. However, local CXL memory accesses (to a memory expander on the same machine) and remote CXL memory accesses (to the memory of another machine) introduce additional latency over DRAM [5], and that latency can have a large impact on performance. Recent work has set out to address these performance concerns in several different ways.

3 Related Work

3.1 Host Memory Expansion via Remote CXL

FaRM builds a shared address space that spans the memory of all of the compute nodes in the cluster over the network via RDMA [6]. Pond [12] moves DRAM into a remote memory pool that is accessed via CXL in order to reduce the amount of idle DRAM in the cluster of machines; this lowers the total amount of DRAM needed across a cluster. Pond combines several ideas to reduce VM DRAM costs by several percent with a small impact on performance. Pond's pools only service a small number of compute nodes (up to 16) in order to reduce the complexity of its network topology, because increasing the complexity of the network topology increases the latency of accesses. Pond uses the memory pools as a place to store infrequently used data. A combination of hardware and software allows Pond to reduce the total amount of needed cluster DRAM without drastically reducing the performance of the applications running on those systems.

Kona advocates for tracking dirty data for remote memory accesses at (64 B) cacheline-granularity in order to reduce the amount of network traffic on dirty data writebacks [4]. It uses local memory as a cache for a larger remote memory pool, and when data is evicted from the local memory cache, the dirty data is written back at a smaller granularity. Memory tiering and cacheline-granular remote memory sharing as with CXL will require rethinking the networking; for example, the overhead of triggering many small remote accesses could limit performance even when network bandwidth is not saturated. Larger remote access granularities could easily be network bandwidth limited. Kona's claims to improve average memory access time by 1.7-5× while reducing dirty data writeback amplification by 2-10×. Both of these benefits are important if memory pooling is going to find practical use.

3.2 Prefetching for Far Memory

Works like 3PO [3], LEAP [17], HoPP [13] and Cache In Hand [10] all explore prefetching for far memory. 3PO uses previous executions to make a trace of memory accesses. LEAP uses heuristics to determine what pages to fetch. Cache In Hand moved prefetching off of the CPU onto CXL devices for CXL-enabled solid-state drives. This allowed it to perform cacheline-granular prefetching for the CPU and inject predictions into the host CPUs' last-level caches. By moving the prefetching off of the CPU, they are able to take advantage of a much larger prefetching model that uses machine learning. HoPP takes a different approach that uses software-based, page-granular prefetching, but it proposes hardware changes in order to get additional information from the memory controller. Additionally, there have been several works that have used machine learning techniques for hardware data prefetchers that operate at the CPU's cache level, these works include Voyager [21] and Pythia [2].

Works like Cache in Hand [10] and HoPP [13] both have a similar motivation to this paper in applying prefetching to disaggregated systems (for solid-state drives and memory respectively). In contrast, this paper specifically explores whether cacheline-granular prefetching makes sense when fetching from a remote CXL device to a local one (and not directly into the host CPU's cache or DRAM), and how it compares to fetching data at page granularity.

4 Comparing Remote Access Approaches

A high coverage cacheline-granular prefetcher could allow for much more efficient use of the cache (in our case, DRAM on the local CXL device), and more efficient use of network bandwidth, since it would move less data per remote access, while still maintaining a high cache hit rate. However, ample cache capacity and network bandwidth might make it more effective to perform remote accesses at a larger granularity. Our goal is to investigate that space. Accessing at a page granularity naturally leverages spatial locality, which comes from frequently accessing items in a common region (a page), so fetching larger regions will more likely result in a hit; however, fetching large regions can pollute the cache with potentially useless data. A goal of this work is to explore the performance of using cacheline-granular prefetching for remote memory and to see if it can be competitive with simpler page granular demand fetching with no prefetching for CXL memory pooling.

To make these comparisons we used a set of common metrics to describe and compare these prefetchers. These metrics are: coverage, accuracy and IPC. Accuracy is the ratio of correct prefetches to the number of issued prefetches. Coverage is the proportion of future memory accesses that a prefetcher is able to cover by issuing prefetches. Unlike with accuracy, coverage doesn't decrease when issuing incorrect prefetches, as it is the ratio of correct prefetches and total memory accesses. This differs from accuracy since it isn't

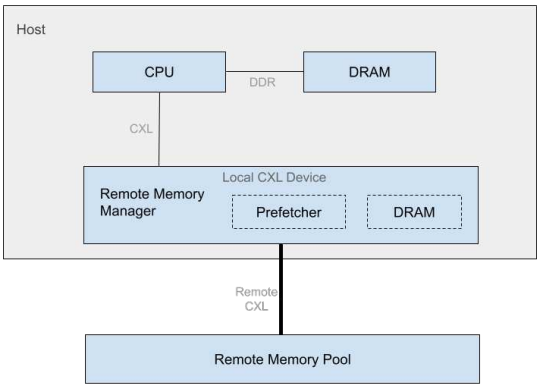


Figure 1: Visual depiction of the targeted setup. A system with a CXL device that has its own local DRAM connected to remote CXL devices with DRAM.

concerned with how many prefetches are issued, only how many times the prefetcher was useful. IPC is the number of Instructions Per Cycle, a high IPC can mean that less time was spent waiting on memory. Seeing a jump in IPC when using a prefetcher points to that prefetcher making correct predictions, and provides a more complete representation of the performance impact of a prefetcher over just looking at accuracy and coverage.

5 Remote Memory Simulation

Like in Cache in Hand [10], our simulated prefetcher operates below the CPUs' last level caches (LLC), meaning that it only sees the accesses that DRAM handle in a typical configuration (Figure 1). This is an important constraint, since deploying such a prefetcher wouldn't require changes to existing CPU hardware; i.e., it is the level of visibility that a CXL device has on a commodity machine. Hence, the access pattern is obscured by the LLC and the cache levels above it. Our traces don't include writes, prefetches or executable code for simplicity; it is common in remote memory systems for stack and executable address space segments to be kept in local memory. We assume that multiple hosts do not operate concurrently on the same remote memory addresses; this is the typical pattern in memory pooling. Shared memory across hosts requires cross-host coherence, which is costly.

Unlike Cache in Hand [10], which prefetches directly into the CPU's cache, our simulator prefetches from a remote CXL device into memory on a local CXL device. Our design simulates on-CXL-device DRAM as a cache with a limited size in order to ensure cache evictions are appropriately triggered during the simulations. Anything not in in the CPU's standard cache hierarchy, local DRAM, or this local CXL-device cache has to go to remote CXL memory, paying a large time penalty. We modeled the time penalty using others' recent reports on CXL [5]. Local CXL hits are 170 ns, whereas remote CXL latencies are 250 ns. We did not model additional delay for a local miss that eventually results in a remote access; the cost of an access missing in the local machine's CXL DRAM cache and being fetched from remote memory is 250 ns.

In addition to comparing cacheline prefetching and page-granular access, we also explored the performance penalty caused by mediating remote access through a local CXL device instead of directly at the local memory controller. Depending on the workload, this additional delay could have a large impact on performance.

We used a modified version of ChampSim that allows prefetches to be generated with a Python script [7]; we also use some of their provided traces and baseline prefetchers. We modified the simulator's DRAM controller so that any reads or reads for ownership of data would go through our simulated CXL device instead of to DRAM. First, on each load that misses at the CPU's LLC the simulator checks to see if the data for the requested address is in the DRAM cached on the local CXL device. If it is not there, a pending remote access queue is checked to see if any previous accesses triggered a prefetch of that address to see if the device is still waiting for it to arrive. Any accesses that get their return time from the queue only pay the remaining latency on the remote request with the lowest amount of latency being 170 ns. If a request has not already been added to the queue, a remote request is issued, and it is added to the queue and the simulator is told to wait 250 ns for that request before moving it into the DRAM cache.

The simulated cache is simple a direct mapped cache; while this might not be representative of state-of-the-art cache systems, it emphasizes the impact of cache thrashing. Other works have explored more sophisticated ways of applying caching techniques to DRAM [1]. Our results also include numbers for fully local DRAM accesses, fully local CXL DRAM accesses, and fully remote CXL DRAM accesses.

There are a few important limitations of our test setup. (1) A small number of instructions limits the amount of memory accessed, which forces us to use smaller cache sizes in order to see differences in behavior. Real systems would be operating at much larger scales. (2) We have a simple cache replacement policy, so important things can be replaced easily due to direct mapping. (3) Row buffer hits and bank occupancy are not fully simulated with the CXL device due to how it is implemented in the DRAM controller. (4) We simulate no local miss cost on remote accesses. A realistic remote access would be slightly higher due to misses in the local CXL DRAM cache taking some time if remote accesses weren't issued speculatively. (5) The simulation only performs latency adjustments on data reads and reads for ownership in order to avoid having executable code on the remote memory. (6) The simulation doesn't account for potential latency increases when accessing at a larger granularity. (7) Flushing data from the LLC into DRAM and from the DRAM cache to the remote memory is not fully modeled, which could have some impact on prefetcher performance since we are not notifying the prefetchers of that. (8) Without having a prefetcher at the LLC (or above) there is a performance cost paid across all of the tests. This was done in part to avoid having the cacheline-granular prefetchers we were testing becoming redundant, and since the predictive data accesses would also affect the page granular results. Further work should be adapted to work with the CPU's prefetcher.

6 Methodology

We tested on subsets of seven data sets across SPEC06, SPEC17 and GAP as provided with the ChampSim fork [7] each with 10 million

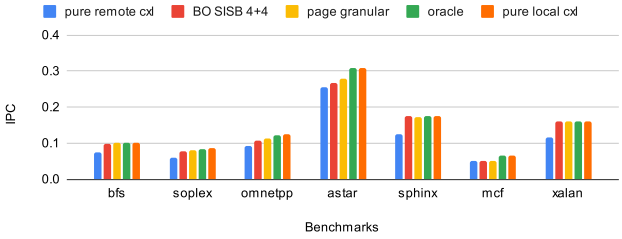


Figure 2: Instructions per cycle (IPC) differences with a small (2 MB) cache in DRAM at the local CXL device.

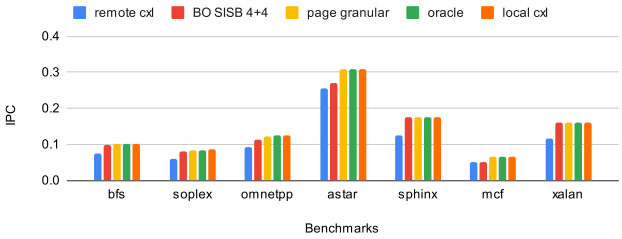


Figure 3: Instructions per cycle (IPC) differences with a large (200 MB) cache in DRAM at the local CXL device.

instructions executed. We ran the following specific benchmarks from these suites:

- **`astar`** 2D path-finding library that is used in game AI [8]
- **`bfs`** Breadth first search [20]
- **`mcf`** Combinatorial optimization [16]
- **`omnetpp`** Discrete Event Simulation [22]
- **`soplex`** Simplex Linear Program (LP) Solver [19]
- **`sphinx3`** Speech Recognition [11]
- **`xlancbmk`** XSLT processor for transforming XML documents into HTML, text, or other XML document types [24]

Our tests initially focus on the impact of prefetch distance, prefetch accuracy, and cache size on performance (IPC). Since these data sets used a limited number of pages we tested with small cache sizes in order to increase the likelihood of cache lines being evicted. Since these benchmarks are not moving large amounts of data, the smaller caches make a noticeable difference in IPC.

Our set up simulates a small amount of DRAM cache space at the CXL device for applications. Figure 2 and Figure 3 show the results for a 2 MB and a 200 MB cache, respectively. Each figure contains bars for several different approaches to prefetching. The oracle was created by playing back an access trace and prefetching a memory address that is 10 accesses in the future. While this isn't a true oracle that knows the optimal values to prefetch, it still serves as a reasonable upper bound estimate for most cases. In addition to this oracle prefetcher, we included combined results from Best Offset [18] and the simplified version of ISB [9] included with the ChampSim fork. Best Offset and Simplified-ISB (SISB) are both given the ability to fetch up to four predictions for every cacheline access that hits the CXL device. The baselines included in the figure are accessing everything from remote memory (*pure remote cxl*), accessing everything from local memory (*pure local cxl*) and accessing at page granularity without prefetching.

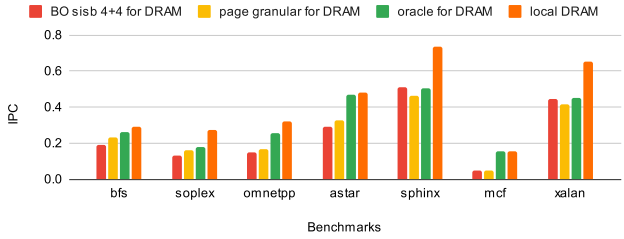


Figure 4: Instructions per cycle (IPC) differences with small (2 MB) cache. Local hits pay CPU DRAM latency rather than the local CXL latency.

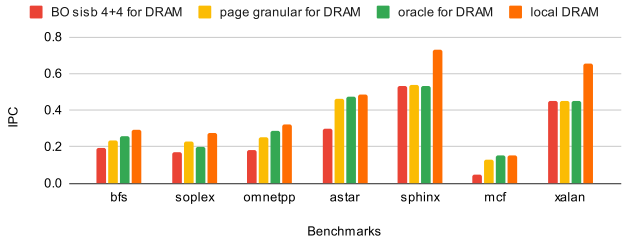


Figure 5: Instructions per cycle (IPC) differences with large (200 MB) cache. Local hits pay CPU DRAM latency rather the local CXL latency.

In addition to testing remote CXL and local CXL, we simulated a set up where the latency for a local hit was that of a DRAM access (Figures 4 and 5). This would be simulating something between HoPP [13] and Cache in Hand [10]. In this simulation, prefetched data is stored in the host CPU's DRAM rather than DRAM on a locally-attached CXL device. However, here, these fetches are at different granularities with/without prefetching.

7 Results and Insights

Here, we highlight the most important insights from these simulations.

(4 KB) pages are hard to beat. First, page granular accesses with no prefetching largely gives equal or better performance than accessing at a cacheline granularity even with future knowledge (compare *oracle* to *page granular* in Figure 2 and 3). For the data sets that we have tested there is sufficient spatial locality so finer-grained caching with prefetching rarely pays off (`omnetpp` and `mcf` are exceptions, likely due to irregular workloads). Prior work showed the performance benefits of prefetching at cacheline granularity in similar contexts [10], but if performance is the primary concern, then page granular accesses, appear to be sufficient in our examples. However, there is a case to be made for cacheline-granular prefetching when operating at DRAM latencies with a small cache size; with larger cache sizes it probably makes sense to pursue prefetching at a page granularity instead of at a cacheline granularity for certain workloads.

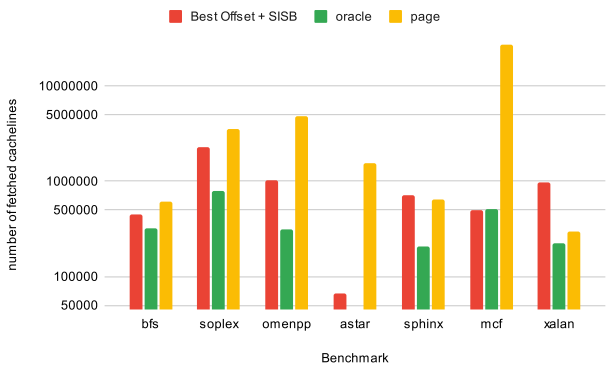


Figure 6: Total number of fetched cachelines per benchmark with the 2 MB cache (logarithmic scale).

Even with small DRAM cache sizes, page granular accesses are sufficient for improving performance. Figure 6 shows the differences in the amount of data fetched from the remote memory. Even though pages increase cache pollution, they also bring some spatial locality benefits. So far our results suggest these effects combine resulting in little difference between the approaches.

Interposing on accesses with CXL has a high cost. These workloads show that there is a large application-level performance penalty when using locally-attached CXL memory compared to DRAM via DDR. For example, comparing the IPCs between Figure 3 where data is cached on the local CXL device to Figure 5 where data is cached in the CPU's local DRAM shows substantial overall IPC improvements.

Today's approaches to remote memory and memory tiering commonly cache remote pages in local DRAM, directly mapping the cached pages using address translation hardware. These results suggest that the additional latency experienced when using CXL will cause significant performance loss. As a result, using CXL to interpose on and track remote memory accesses would need to be combined with an approach that caches the hottest pages or cachelines in local DRAM in order to be effective [27].

Remote access via CXL does not drastically impact performance over access to local CXL memory. Counterintuitively, based on estimated remote CXL costs, the additional relative overhead of accessing remote memory over accessing DRAM attached via local CXL is low. Figures 2 and 3 show this as the differences between the *pure local cxl* and *pure remote cxl* bars. If this performance penalty is acceptable, then local CXL DRAM caches don't make much sense since purely remote accesses get close to the performance of accessing local DRAM via CXL in the benchmarks that we tested.

Adaptive access granularity may work well for remote memory. While our tests didn't explore the cost of writing dirty data back to the remote memory, Figure 6 shows that cacheline-granular accesses, even with prefetching, can greatly reduce the amount of data being fetched for certain benchmarks. However, as previously mentioned, accessing at a larger granularity gets performance close to fully local CXL accesses in many cases. This suggests that there is a trade-off of accuracy and granularity. If a prefetcher has high accuracy and coverage for an application, then accesses can be at a small granularity. However, if the prefetcher has low accuracy and low coverage, then accesses should be adjusted in order to take advantage of any potential spatial locality. Further investigation into optimal adjustable access granularity, similar to [23] could potentially lead to combining prefetchers and adjustable fetch sizes together in order to optimize for workloads. These systems would also have to take into account the size of the cache, since that also plays a part in performance. Systems with a large amount of contention for cache capacity or for network bandwidth could take more advantage of smaller granularity prefetchers, whereas systems with a small amount of contention probably wouldn't see any benefit over using page granular accesses.

8 Conclusion

CXL interested us in new designs for remote memory prefetching. It seems like a natural fit for prefetching since, for the first time, CPUs can expose coherence events in a micro-architecture-agnostic way; however, our explorations here have tempered our expectations, and, as a result, we have adapted our focus in several key ways.

Our simulations show that leveraging of spatial locality within pages (at least for the subsets of these workloads) is sufficient to get close performance to fully local CXL accesses and local DRAM accesses. This means that, at the moment, it is tough for prefetching at cacheline granularity to provide a benefit over just fetching at page granularity in our experiments.

Next, our results show that the subsets of these benchmarks are sensitive to the initial jump from host-attached DRAM to local CXL DRAM, and they are less sensitive when jumping from local CXL DRAM to remote CXL DRAM. This supports the direction of works like Pond [12], where there isn't an additional local CXL device between the CPU and the pooled remote memory and where frequent use of remote memory is avoided. Further investigations into moving memory between CXL DRAM and host CPU DRAM could open up new research questions and potentially better use of CXL memory pools.

In the future, remote memory prefetching and local caching might be managed at the host CPU memory controller level instead of via CXL devices to avoid paying additional latency from going through a CXL bus. This would provide the benefits of directly local DRAM cache lines cached cache lines or pages while still providing the scheduling and memory allocation flexibility of remote memory pools.

In conclusion, our limited explorations don't necessarily "close the book" on cacheline-granularity prefetching for remote memory, but it is causing us to adjust our approach to it. CXL remote memory presents an interesting set of challenges in terms of what problems need to be solved and where it can make a practical impact on cost and performance in real-world workloads.

Acknowledgements

Thank you to Prof. Vijay Nagarajan for helping when this was a class project. Thank you to David Dursteler and Sweta Ukey for helping with explorations that motivated this work.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2245999. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This work was also supported in part by VMware.

References

[1] Maryam Babaie, Ayaz Akram, Wendy Elsasser, Brent Haukness, Michael Miller, Taeksang Song, Thomas Vogelsang, Steven Woo, and Jason Lowe-Power. 2024. TDRAM: Tag-enhanced DRAM for Efficient Caching. (2024). doi:10.48550/arXiv.2404.14617 arXiv:2404.14617 [cs.AR]

[2] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. *MICRO* (2021).

[3] Christopher Branner-Augmon, Narek Galstyan, Sam Kumar, Emmanuel Amaro, Amy Ousterhout, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2022. 3PO: Programmed Far-Memory Prefetching for Oblivious Applications. (2022).

[4] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. 2021. Rethinking software runtimes for disaggregated memory. *ASPLOS* (2021).

[5] Daniel S. Berger Debendra Das Sharma, Robert Blankenship. 2024. An Introduction to the Compute Express LinkTM (CXLTM) Interconnec.

[6] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, , and Miguel Castro. 2014. FaRM: Fast Remote Memory. *NSDI 14* (2014).

[7] Quang Duong. [n. d.]. https://github.com/Quangmire/ChampSim.

[8] Lev Dymchenko. 2006. astar. *https://www.spec.org/cpu2006/Docs/473.astar.html* (2006).

[9] Akanksha Jain and Calvin Lin. 2013. Linearizing irregular memory accesses for improved correlated prefetching. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 247–259.

[10] Miryeong Kwon, Sangwon Lee, and Myoungsoo Jung. 2023. Cache in Hand: Expander-Driven CXL Prefetcher for Next Generation CXL-SSDs. (2023).

[11] Sphinx Speech Group Carnegie Mellon University Evandro Gouevea Arthur Chan Richard Stern Paul Lamere. 2006. sphinx. *https://www.spec.org/cpu2006/Docs/482.sphinx3.html* (2006).

[12] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 574–587. doi:10.1145/3575693.3578835

[13] Haifeng Li, Ke Liu, Ting Liang, Zuojun Li, Tianyue Lu, Hui Yuan, Yinben Xia, Yungang Bao, Mingyu Chen, and Yizhou Shan. 2023. HoPP: Hardware-Software Co-Designed Page Prefetching for Disaggregated Memory. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1168–1181. doi:10.1109/HPCA56546.2023.10070986

[14] Kai Li and Paul Hudak. 1989. Memory coherence in shared virtual memory systems. *ACM Trans. Comput. Syst.* 7, 4 (Nov. 1989), 321–359. doi:10.1145/75104.75105

[15] Parthasarathy Ranganathan Luiz André Barroso, Urs Hölzle. 2019. *The Datacenter as a Computer*. Springer.

[16] Andreas Löbel. 2006. mcf. *https://www.spec.org/cpu2006/Docs/429.mcf.html* (2006).

[17] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively Prefetching Remote Memory with Leap. (2020).

[18] Pierre Michaud. 2016. Best-offset hardware prefetching. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 469–480. doi:10.1109/HPCA.2016.7446087

[19] Tobias Achterberg Roland Wunderling, Thorsten Koch. 2006. soplex. *https://www.spec.org/cpu2006/Docs/450.soplex.html* (2006).

[20] David Patterson Scott Beamer, Krste Asanović. 2015. The GAP Benchmark Suite. *arXiv:1508.03619 [cs.DC] https://github.com/sbeamer/gapbs* (2015).

[21] Zhan Shi, Akanksha Jain, Milad Hashemi Kevin Swersky, Parthasarathy Ranganathan, and Calvin Lin. 2021. A hierarchical neural model of data prefetching. *ASPLOS* (2021).

[22] András Varga. 2006. omnetpp. *https://www.spec.org/cpu2006/Docs/471.omnetpp.html* (2006).

[23] Alexander V. Veidenbaum, Weiyu Tang, Rajesh Gupta, Alexandru Nicolau, and Xiaomei Ji. 1999. Adapting cache line size to application behavior. (1999).

[24] IBM Corporation Apache Inc Christopher Cambly Andrew Godbout Neil Graham Sasha Kasapinovic Jim McInnes June Ng Michael Wong. 2006. xalan. *https://www.spec.org/cpu2006/Docs/483.xalancbmk.html* (2006).

[25] Mingxing Zhang, Teng Ma, Jinqi Hua, Zheng Liu, Kang Chen, Ning Ding, Fan Du, Jinlei Jiang, Tao Ma, and Yongwei Wu. 2023. Partial Failure Resilient Memory Management System for (CXL-based) Distributed Shared Memory. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 658–674. doi:10.1145/3600006.3613135

[26] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing Memory Tiers with CXL in Virtualized Environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 37–56. https://www.usenix.org/conference/osdi24/presentation/zhong-yuhong

[27] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing Memory Tiers with CXL in Virtualized Environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 37–56. https://www.usenix.org/conference/osdi24/presentation/zhong-yuhong