Prefetching in CXL-based Server Processors

Srikar Vanavasam

Abstract

The memory subsystems of modern server processors have undergone significant architectural changes in recent years. Furthermore, emerging technologies such as CXL are poised to become widely adopted in the industry. This work studies how data prefetching in server processors is impacted by these trends and in particular focuses on latency-critical datacenter applications. We show that state-of-the-art data prefetchers are not effective in bandwidth-contained configurations that server processors typically operate. However, we show that data prefetching becomes more important in a CXL-based processor due to the increased memory bandwidth and increased memory access latency.

Approved By:

Dr. Alexandros Daglis
School of Computer Science
Georgia Institute of Technology

Dr. Moinuddin Qureshi
School of Computer Science
Georgia Institute of Technology

1 Introduction

Datacenters are the workhorses of the modern digital economy. They must support various workloads, including data analytics, web search, and machine learning. Many of these workloads place a considerable load on the memory subsystem, which can significantly influence overall performance. Recent trends such as increased processor core counts, larger code footprints, and bigger datasets further increase the pressure on server memory systems. In response, CPUs now include ever bigger caches, including large L3 victim caches, in order to capture large working sets. Modern processors also make use of hardware data prefetching in order to reduce the number of compulsory cache misses and significantly improve overall performance. However, prior works proposing state-of-the-art prefetchers have not evaluated the prefetchers in the context of modern server processors and workloads. Additionally, emerging technologies offer a wide range of performance trade-offs within the memory system. In particular, Compute Express Link (CXL) is an emerging serial interconnect that offers significantly higher bandwidth compared to DDR at the expense of increased latency. This work studies the performance of Bingo, [1] a state-of-the-art prefetcher, in server processors and its potential in CXL-based memory systems.

The effectiveness of prefetching is highly dependent on the memory access pattern of the workload, and as such, different classes of workloads vary greatly in the benefits they see from prefetching, as shown in prior works. Latency-critical applications are increasingly common in datacenters [2] but are not well represented in prior studies. This work focuses on evaluating datacenter workloads from the Tailbench benchmark suite [2]. Additionally, prior works do not reflect changes that have occurred in server memory systems in recent years. Notably, server processors now include large private caches and smaller non-inclusive LLCs in order to reduce data duplication in the cache hierarchy. This study will reflect these changes in server processors to produce a modern evaluation of prefetching in server processors.

2 Related Works

A variety of prior work is relevant to this project. Several works propose state-of-the-art prefetchers. There are also benchmark suites and applications that are representative of datacenter workloads. Finally, there are existing simulation tools that will be used to perform this study.

2.1 Prefetchers

Bingo [1] is a recent proposal for spatial data prefetching, as well as the runner-up of the third Data Prefetching Championship. Bingo is based on the observation that events either are short or long. Short events, such as PC+Offset, have a high probability of recurrence but are less accurate. On the other hand, long events, such as PC+Address, result in high accuracy prefetches but have lower miss coverage since the probability of events recurring is lower. As such, Bingo proposes associating the observed footprint information to multiple events and using the longest matching event in order to provide both high opportunity and high accuracy.

Pythia [3] is another recent data prefetcher proposal that formulates the prefetcher as a reinforcement learning agent. For every demand request, Pythia observes multiple different types of program context information to make a prefetch decision. For every prefetch decision, Pythia receives a numerical reward that evaluates prefetch quality under the current memory bandwidth usage. This enables Pythia to correlate program context with prefetch decisions and results in accurate and timely prefetches.

In this work, we focus our evaluation on Bingo as it is representative of state-of-the-art in data prefetching and is more feasible to adapt to our simulation infrastructure and workloads than other proposed prefetchers.

2.2 Benchmark Suites

Prior work has provided a collection of applications and benchmarks that are representative of datacenter workloads that can be characterized. One such benchmark suite is CloudSuite [4], which focuses on scale-out workloads that includes many common datacenter applications such as web search, streaming, and MapReduce. Similarly, DCBench [5] is another benchmark suite and aims to be representative of datacenter workloads by including several data analysis workloads. In this work, we focus on Tailbench [2], which represents common datacenter applications that must maintain a small and predictable tail latency. Such applications, including web search, machine learning inference, and key-value stores, are increasingly common in the datacenter and are well represented in Tailbench. Additionally, several of these workloads are bandwidth sensitive and can experience significant performance degradation from prefetcher overprediction under tight bandwidth constraints which have not been deeply explored in prior prefetcher works.

2.3 Simulation Infrastructure

Finally, prior work has provided the means to efficiently and accurately evaluate architectural changes through the use of simulators. ZSim [6] is an instrumentation-based microarchitectural simulator that can simulate many core systems in a highly parallel manner by using a two-phase parallel simulation that leverages dynamic binary translation. This allows ZSim to simulate chips with hundreds or even thousands of out-of-order cores at a reasonable speed and accuracy, making it an ideal choice for the simulation of many core server CPUs. Another simulator is DRAMSim2 [7], which accurately models the internal details of a DRAM memory system and offers a simple programming interface that can be configured to model a range of memory systems. Although ZSim provides a custom memory model that promises faster simulation, DRAMSim2 provides a more accurate memory model, which is crucial in investigating memory systems.

3 Methodology

3.1 System Configurations

We use ZSim [6], a fast and accurate microarchitectural simulator, to model a typical server processor, and DRAMSim2 [7] to model a typical DDR5 memory system. The baseline configuration for the simulation is loosely modeled after the AMD EPYC Genoa architecture [8], which is detailed in Table 1. To model CXL-based systems, we consider configurations of up to 5 memory controllers or 2.5x more bandwidth than the baseline system and with 20ns-60ns memory access latency overhead. This level of bandwidth increase from CXL is a conservative estimate based on the increased pin-efficiency of CXL, which enables greater bandwidth while using the same number of CPU pins. Additionally, the 20ns-60ns memory access latency overhead over DDR is expected to be achieved by production CXL systems [9].

CPU	16 OoO cores, 2GHz, 4-wide, 256-entry ROB
L1	32KB L1-I & L1-D, 8-way, 64B blocks, 4-cycle access
L2	1MB, 8-way, 12-cycle access
LLC	2 MB/core, shared & non-inclusive, 16-way, 46-cycle access
Memory	DDR5-4800, 128 GB per channel, 2 sub-channels per channel, 2 ranks per sub-channel, 16 banks per rank **DDR baseline:** 2 memory controllers **CXL baseline:** 4 memory controllers

Table 1: Baseline system configuration for Zsim.

3.2 Workloads

We evaluate applications found in datacenter environments. We deploy the same workload instance on all cores and simulate 200 million instructions per core after fast-forwarding each application to a region of interest. We use five applications from the Tailbench[2] benchmark suite with their default datasets including: search engine (xapian), B+-tree-based key-value store (masstree), real-time translation (moses), speech recognition (sphinx), and image recognition (imgdnn).

3.3 Prefetcher

We evaluate Bingo [1], which we implement in ZSim based on source code provided by the authors. Adapting ZSim to support prefetchers such as Bingo required modifications to ZSim's cache coherence implementation and out-of-order core model. Table 2 shows the specific parameters used for Bingo. The region size parameter is the granularity at which Bingo tracks memory access patterns. The value of the region size was chosen empirically by the authors of Bingo to be large enough to capture most memory access patterns. The filter table (FT), accumulation table (AT), and pattern history table (PHT) track current access histories. The PHT is primarily where pattern histories are stored, and as such, we choose a PHT size beyond the size at which prefetcher coverage plateaus to ensure prefetcher performance is not limited by storage. The prefetcher is trained on L1-cache misses and fills prefetched lines directly into the L2 bypassing the LLC.

Region Size	2KB
FT	64 entries
PHT	64K entries
AT	128 entries

Table 2: Bingo Parameters

4 Evaluation

4.1 Miss Coverage and Overprediction

To evaluate the effectiveness of the Bingo prefetcher in our setup, Figure 1 shows the coverage and overprediction for a microbenchmark and each of the evaluated workloads. Covered misses are the ones that are successfully captured by a prefetcher. Overpredictions are incorrect prefetches, which are normalized to the number of data misses in the baseline system without a prefetcher. However, our coverage metric does not account for the timeliness of the prefetch, only whether a prefetch was issued for a cache block before it is demanded by the core. To validate the correctness of our Bingo implementation, we examine a simple array traversal benchmark. The microbenchmark traverses the array in an easy to predict stride pattern, which the prefetcher easily recognizes, leading to a perfect miss coverage rate and no overprediction.

The miss coverage rates of the examined workloads are lower than those of the workloads studied in the original Bingo paper. This is to be expected as the server workloads shown here tend to have more complex access patterns that cannot easily be captured by the prefetcher, which has been supported by prior works [4]. However, the workloads shown also do not suffer from a high degree of overprediction. A likely explanation for this is that complex access patterns of the workloads do not generate many matching events in Bingo's history table, leading to fewer overpredictions. The low degree of overpredictions for these workloads suggests the prefetcher should be effective even in bandwidth-constrained environments as few unnecessary cache lines are fetched, leading to little wasted bandwidth.

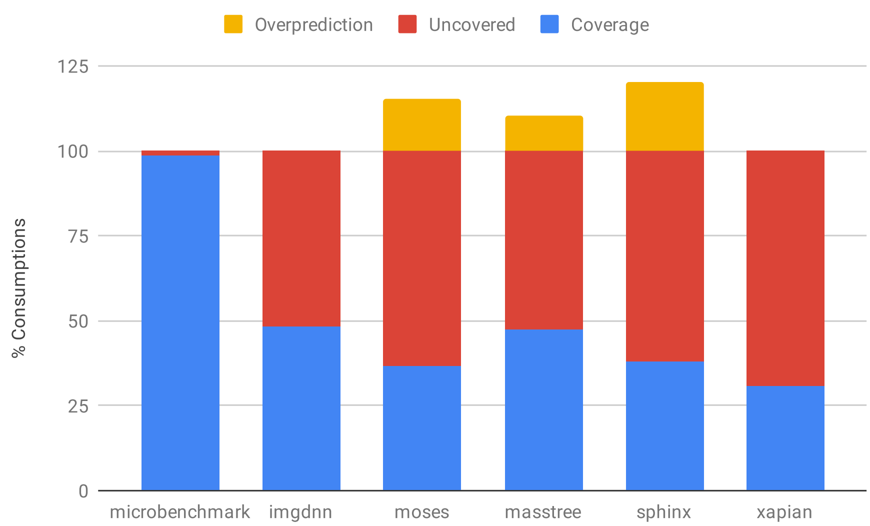

Figure 1: Coverage and overprediction of Bingo.

4.2 Bandwidth Sensitivity

To evaluate the prefetcher in bandwidth-constrained environments that are found in modern server processors, we examine the speedup offered by the prefetcher at various levels of bandwidth as shown in Figure 2. We simply increase and decrease the number of memory controllers in the system to scale the bandwidth. In general, speedup increases with bandwidth, and at low bandwidths, several workloads experience a slowdown. This is due to two effects: reduced latency impact of overpredictions and increased timeliness of prefetches. Overpredictions lead to unnecessary cache lines being fetched. This both pollutes the cache and uses more memory bandwidth. Increased memory bandwidth usage leads to greater memory access latency due to increased queuing delays in the memory controller. This is why moses, masstree, and sphinx, the workloads with significant overprediction, experience slowdown from the prefetcher at low memory bandwidths. However, at greater bandwidths, the impact of overprediction is lower and the workloads with overprediction see a speedup.

The other significant effect of memory bandwidth on prefetcher speedup is the timeliness of prefetches. At low bandwidths, the increased memory latency due to queuing makes prefetches less timely. As prefetches take longer to retrieve the required cache line, it is more likely that the core will issue a demand access for the line before the line is brought into the cache by the prefetch request.

This increases the latency experienced for the demand access despite the fact that the prefetcher correctly issued the prefetch. This is why applications that do not experience overprediction still see increases in speedup as bandwidth increases. Imgdnn in particular experiences a significantly higher speedup than the other applications. This is due to its high bandwidth utilization, as seen in Figure 3, which makes the aforementioned effects more significant.

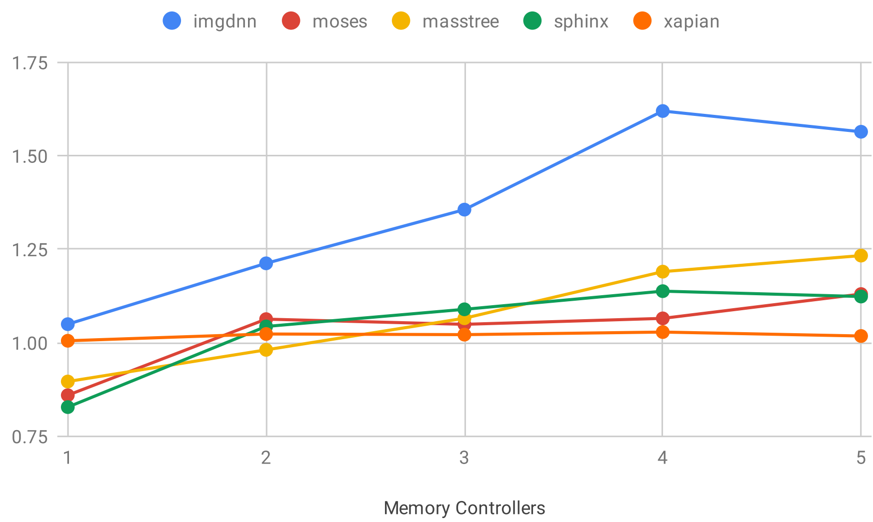

Figure 2: Performance improvement with varying memory bandwidth (memory controllers).

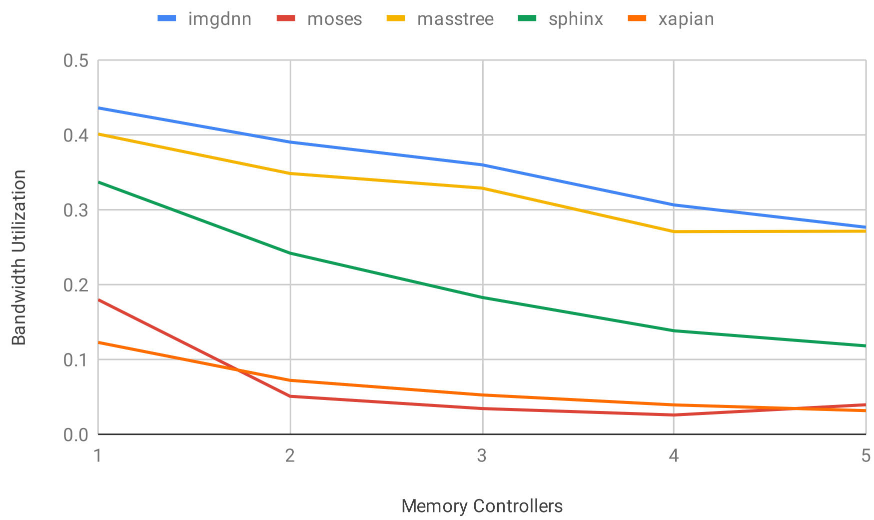

Figure 3: Memory bandwidth utilization with varying memory controllers.

4.3 CXL Delay

Figure 2 examines memory bandwidth up to a ratio of 5 memory controllers to 16 cores. However, this is likely beyond what can be achieved with DDR in server processors as current server processors already exceed a 1:8 ratio of memory controllers to cores [10]. As such, we examine these high memory bandwidth design points in the context of CXL, which enables these higher bandwidths but at the expense of higher memory latency. Figures 4 and 5 show the speedup of the baseline CXL system (4 memory controllers) over the baseline DDR system (2 memory controllers) with varying CXL latency overhead. As figure 4 shows, the CXL-based system without prefetching generally performs worse than the DDR system without prefetching due to the latency overhead, and as expected, the slowdown increases with the higher latency overhead. However, as figure 5 shows, with prefetching enabled on both systems, the CXL-based system offers significant speedups over DDR for most of the applications. This is because the prefetcher is able to effectively lower the cache miss rate of demand accesses, which reduces the impact of the CXL-based system's memory latency overhead. The lowered impact of the latency overhead then allows the application to benefit from the greater bandwidth availability.

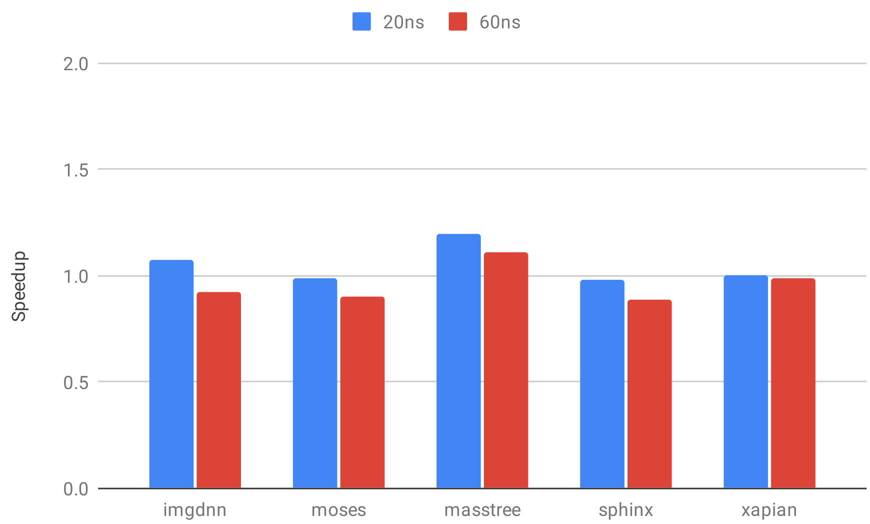

Figure 4: Speedup of CXL over DDR without prefetching and varying CXL latency.

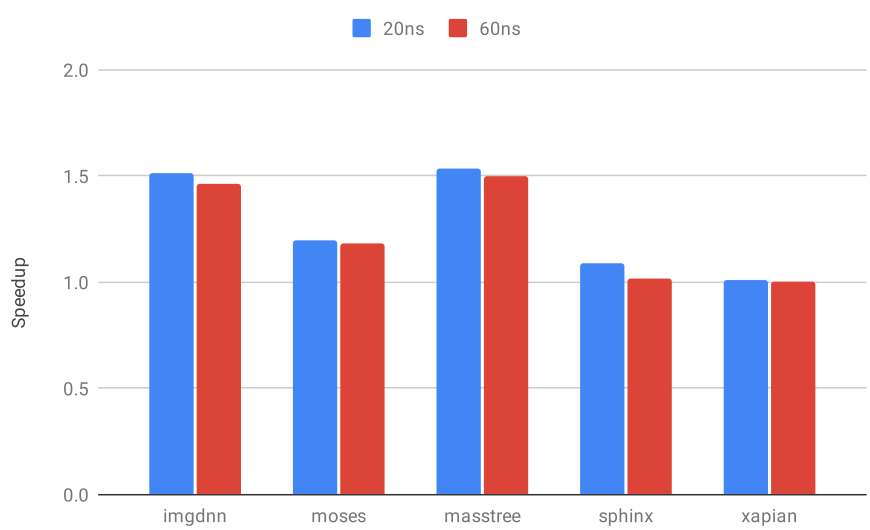

Figure 5: Speedup of CXL over DDR with prefetching and varying CXL latency.

5 Discussion

From the analysis of typical datacenter workloads, we find that prefetching does not provide significant improvement in bandwidth-constrained systems but can provide significant speedups when bandwidth is plentiful. This is further amplified in CXL-based systems as the increased memory bandwidth also comes with added memory latency, which increases the benefit of prefetching.

5.1 DDR-based systems

We examined the impact of prefetching across a range of main memory bandwidths. This range is partitioned into bandwidths feasible in DDR-based systems and those that require a higher bandwidth interface such as CXL. Prior works have generally only considered DDR-based systems, however, the workloads typically examined in works proposing state-of-the-art prefetchers are not as bandwidth constrained as many datacenter workloads. This allows state-of-the-art prefetchers to overpredict and consume additional bandwidth, which is not an issue for non-bandwidth-intensive applications but can lead to substantial performance degradation for bandwidth-intensive applications. This is due to the steep increase in memory access latency when memory bandwidth utilization increases, which is predominantly caused by queuing delays in the memory controller. Due to these effects, bandwidth-intensive datacenter applications do not see a significant performance gain from state-of-the-art prefetchers in bandwidth-constrained DDR-based systems.

5.2 CXL-based systems

Unlike DDR, CXL is a serial interface that allows for much better frequency scaling and better bandwidth per CPU pin. This means that a typical server processor that uses CXL to attach off-chip DRAM instead of DDR will have greater bandwidth availability. However, due to the serialization and deserialization overheads of CXL, there is a latency overhead compared to DDR. We find that this increases the benefit of prefetching as the overhead of missing in the caches is significantly higher and the additional bandwidth reduces the impact of overprediction. These factors make prefetching critical to achieving significant performance benefits in a CXL-based system.

5.3 Future Work

This work examines how prefetcher performance extends to CXL-based memory systems and shows the importance of effective prefetching in such systems. However, there are still unanswered questions that can be addressed in future works. A limitation of this study is that die area was not considered. The evaluated bandwidth improvements from CXL we considered are based on utilizing the same number of pins as a DDR-based processor, however, a CXL memory controller, which is based on PCIe controller, requires more die area than a DDR memory controller. This complicates the implementation of a fully CXL processor as it would require significantly more die area, which may make such a design infeasible. Designing an iso-area CXL processor would require reducing some other components of the processor to make space for the larger CXL controllers. One solution could be to reduce the LLC as doing so is already a tradeoff made by modern server processors. However, the longer memory latency of CXL could benefit from a larger LLC to temporarily buffer the prefetched data. Investigating this further is a promising future work that can help address the feasibility of building a CXL-only processor.

References

[1] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 399–411. DOI: 10.1109/HPCA.2019.00053.

[2] H. Kasture and D. Sanchez, "Tailbench: A benchmark suite and evaluation methodology for latency-critical applications," en, in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, Providence, RI, USA: IEEE, Sep. 2016, pp. 1–10, ISBN: 978-1-5090-3896-1. DOI: 10.1109/IISWC.2016.7581261. [Online]. Available: http://ieeexplore.ieee.org/document/7581261/ (visited on 12/10/2022).

[3] R. Bera, K. Kanellopoulos, A. V. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[4] M. Ferdman, A. Adileh, O. Kocberber, *et al.*, "A Study of Emerging Scale-out Workloads on Modern Hardware," en, p. 11, 2012.

[5] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, Sep. 2013, pp. 66–76. DOI: 10.1109/IISWC.2013.6704671.

[6] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," en, p. 12,

[7] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," en, *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan. 2011, ISSN: 1556-6056. DOI: 10.1109/L-CA.2011.4. [Online]. Available: http://ieeexplore.ieee.org/document/5732229/ (visited on 12/10/2022).

[8] StorageReview, "4th Gen AMD EPYC Review (AMD Genoa)," [Online]. Available: https://www.storagereview.com/review/4th-gen-amd-epyc-review-amd-genoa.

[9] CXL Consortium, *Compute express link™ 3.0 white paper*. [Online]. Available: https://www.computeexpresslink.org/_files/ugd/0c1418_a8713008916044ae9604405d10a7773b.pdf/.

[10] Wccftech, "Amd epyc bergamo 'zen 4c' cpus," [Online]. Available: https://wccftech.com/amd-epyc-bergamo-zen-4c-cpus-being-deployed-in-1h-2023-tackle-arm-instinct-mi300-apu-back-in-labs/amp/.