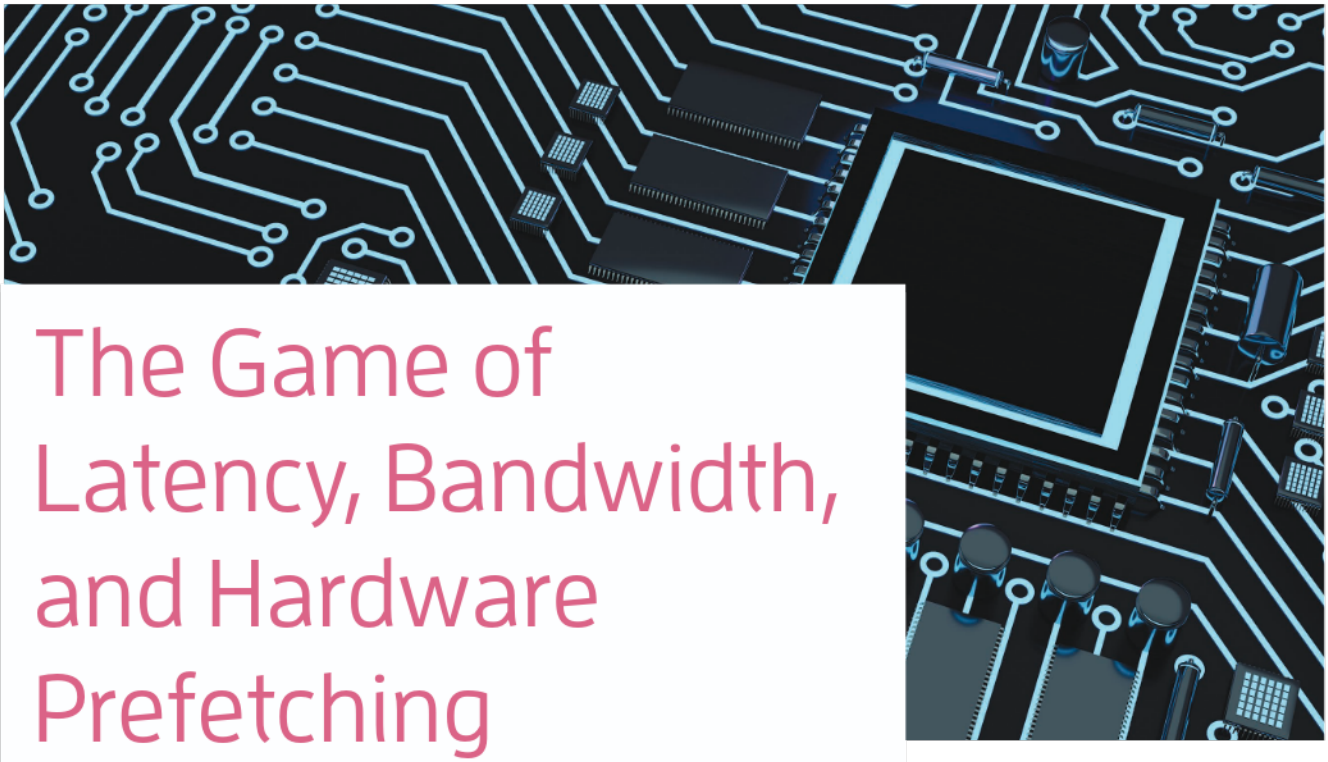

The Game of Latency, Bandwidth, and Hardware Prefetching

Biswabandan Panda, Indian Institute of Technology

A processor's cache hierarchy exploits locality in memory accesses to reduce latency but can't satisfy all memory accesses. Modern processors contain hardware prefetchers to predict data to be used in the future and bring them into a cache in a timely manner.

The performance of modern processors is driven by two key metrics: *latency* and *bandwidth*. In general, the latency and bandwidth of off-chip memory dictate the performance of processors as off-chip memory is slow and fails to deliver data at the rate that the processor demands. In short, a processor wants to READ/WRITE data from/into the memory, and there is no free lunch associated with it. The penalty is the memory access time, which is in hundreds of cycles. In the field of computer architecture, this is termed the *memory wall*. Figure 1 illustrates the concept of a memory wall in the form of memory access latency.

There is an old network saying: "Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed—you can't bribe God."—Anonymous

On-chip caches break the memory wall up to a certain extent by speculating about locality in memory accesses. Multiple levels of cache (typically L1, L2, and L3 in high-performance and mobile systems) in a cache hierarchy bring data closer to the processor and result in a reasonable latency. However, caches do not provide a 100% hit rate, which means that only some of the data a program needs are available at the L1, L2, or L3. What next?

WELCOME TO THE WORLD OF HARDWARE PREFETCHERS

Prefetchers are close friends of the cache hierarchy that bring data into a cache before the processor demands them

Digital Object Identifier 10.1109/MC.2024.3384851
Date of current version: 3 June 2024

0018-9162/24©2024IEEE

EDITORS
TIMOTHY JONES University of Cambridge, U.K., timothy.jones@cl.cam.ac.uk
ROBERT MULLINS University of Cambridge, U.K., robert.mullins@cl.cam.ac.uk

so that the processor will get the data from caches with cache latency and not dynamic RAM (DRAM) latency. At a 10K-foot view, a hardware prefetcher learns memory access patterns by observing past accesses, and based on the learning, it prefetches the data into the cache corresponding to future memory addresses of interest. Figure 2 shows a simple prefetcher with a one-level cache that looks at past accesses (X, X + 1, X + 2) and predicts future accesses (X + 3). Prefetching can be done on the software side too, thanks to intelligent compilers and programmers who can insert prefetch instructions that, when executed by the processor, bring data into the cache.[1]

TWO KNOBS OF INTEREST

Two knobs that drive the effectiveness of a prefetcher are 1) prefetch degree and 2) prefetch distance or depth. To judge the effectiveness of a prefetcher, we use accuracy and coverage. A prefetcher with high coverage leads to better performance as it converts more cache misses into hits. Along the same lines, a prefetcher with high accuracy ensures that valuable data come to the cache without creating additional memory traffic. In an ideal scenario, a prefetcher with 100% accuracy and coverage can fully mitigate the memory wall problem. However, such a prefetcher does not exist, and nearly all prefetcher designs struggle to achieve a balance between these two. The combination of prefetch degree and distance defines the aggressiveness of a prefetcher, which in turn affects its accuracy and coverage. A highly aggressive prefetcher would prefetch a large number of addresses that will be used far ahead in time to the current address stream generated by the processor. Further, a prefetcher with good accuracy and coverage can only perform well if the prefetch response is timely, meaning that the data arrive in the cache just before they are needed. Hence, timeliness plays a crucial role in affecting the overall coverage of the prefetcher.

FIGURE 1. The memory wall problem.

PREDICTING MEMORY ADDRESSES

A simple yet effective hardware prefetcher that is used in commercial processors is known as the *next-line prefetcher*, which does not predict future accesses but assumes that after the processor has accessed a cache line, it will send a request to the next cache line address. This prefetcher does not incur any storage overhead. However, it cannot predict nonlinear memory access patterns. Another well-known prefetcher that is also there in commercial machines is called the *IP-stride prefetcher*, where a prefetcher tries to predict the stride between two memory accesses generated by the same instruction pointer (IP). Figure 3 shows how an IP-stride prefetcher captures the strides between two accesses from one IP. This prefetcher appeared in the early 1990s, and since then, many prefetching techniques have been proposed, which the industry adopted in different avatars.

The architecture community has organized three data prefetching championships at flagship architecture conferences, which has helped push the limits of hardware prefetching. The last championship on data prefetching was held with the International Symposium on Computer Architecture

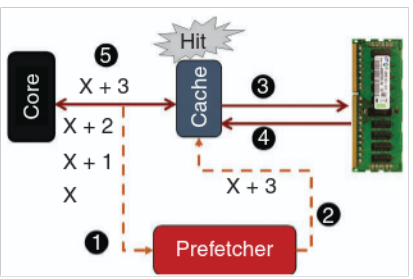

FIGURE 2. A 10K-foot view of a hardware prefetcher with one level of cache.

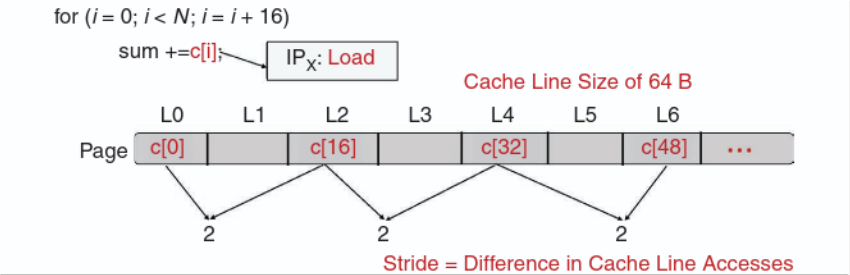

FIGURE 3. An IP-stride prefetcher learning strides in memory accesses.

in 2019.[2] State-of-the-art hardware prefetchers improve performance by more than 10% (as shown in Figure 4) and as high as 2× when compared to the IP-stride prefetcher. State-of-the-art prefetchers try different hardware signatures that can help in learning memory access patterns. For example, IP-stride uses the IP or program counter as the signature. Similarly, different features that can better capture the control and data flow of a program are used to predict future accesses. Depending on the signature used, the storage overhead of prefetchers varies between a few kilobytes to tens of kilobytes.

For workloads with huge memory footprints that thrash both TLBs and caches, TLB prefetching can improve the performance significantly.

THE GOOD, THE BAD, AND THE UGLY

In general, memory access patterns can be classified in three ways. First, there are regular ones that are highly predictable, thanks to the usage of loops in our programs. Second, irregular ones demand intelligent signatures to capture the access patterns. The third, and most difficult, are accesses coming from pointer-chasing codes and indirect memory accesses, such as A[B[C[i]]], where a prefetcher needs to prefetch based on the load values of C[i] and B[C[i]].

Different cache levels see different memory access patterns. The L1 cache is closer to the processor and sees unfiltered memory access patterns, whereas the L2 and L3 caches see filtered memory access patterns based on accesses that miss at the L1. In general, commercial processors use multiple prefetchers at different levels of caches targeting different memory access patterns. Therefore, coordination among these prefetchers is extremely important, or else these prefetchers may interact negatively, nullifying all of the benefits from individual prefetchers. All of these prefetchers work at different aggressiveness levels, so that overall they work in synergy.[3]

So far, we have discussed prefetching issues assuming there is a single core system with multiple levels of caches. However, all modern systems are multicore systems. Do prefetchers work even on multicore systems?

PREFETCHING IN MANY-CORE SYSTEMS

As multicore and many-core systems are commonly used in servers, prefetchers of one core interfere with prefetchers of other cores for the shared resources, like large last-level cache and off-chip DRAM bandwidth. Historically, it has been observed that the DRAM bandwidth does not increase linearly with the increase in core count. Effectively, this bandwidth problem manifests itself into a latency problem and starts hurting the performance of all of the cores in a many-core system. *Prefetch filtering*[5] and *criticality-based prefetching*[6] are two approaches that are effective in many-core systems as these techniques reduce the prefetch traffic coming from prefetch requests that do not contribute much to the overall system performance. Where else can hardware prefetching be used?

Prefetching virtual address translations

So far we have discussed hardware data prefetching that prefetches data into caches. However, this idea of prefetching can be extended to translation look-aside buffers (TLBs) that prefetch virtual-to-physical address translations into the TLB. Modern processors use multiple levels of TLBs that cache the virtual-to-physical address translations. Even though TLB prefetching is not deployed as often as data prefetching, for workloads with huge memory footprints that thrash both TLBs and caches, TLB prefetching can improve the performance significantly. Note that designing a TLB prefetcher is more challenging compared to designing a data-cache prefetcher as the access patterns across operating system (OS) pages are completely different from the access patterns that go to cache lines within an OS page.

Prefetching code

Modern data centers and client/server workloads consist of large code footprints as compared to data footprints, so the need for instruction prefetching is inevitable. In the case of instruction prefetching, a prefetcher prefetches instructions into the L1I cache so that the processor does not stall while fetching instructions. One of the key challenges in designing the

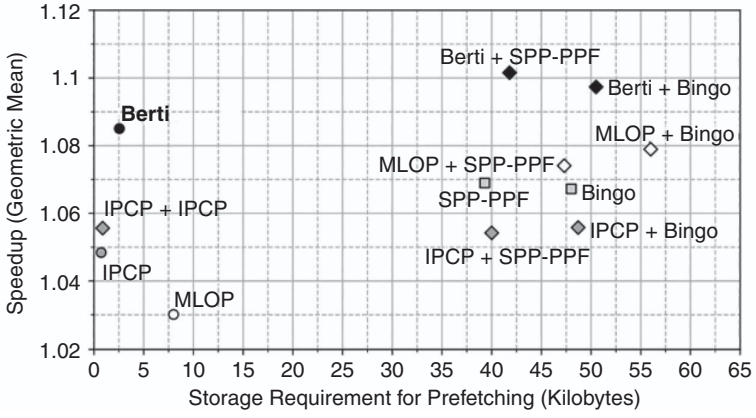

FIGURE 4. Performance of state-of-the-art prefetchers compared to that of the IP-stride prefetcher.[4] X + Y refers to prefetcher X at L1 and prefetcher Y at L2.

instruction prefetcher is to predict the instructions based on the performance of branch predictors and their supporting structures, like branch target buffers (BTBs). As the outcome of branch predictors decides the flow of instructions that should be fetched, an instruction prefetcher uses this information to prefetch in an idea called *fetch-directed instruction prefetching* (*FDIP*), as shown in Figure 5. It is thus intuitive to note that the effectiveness of the FDIP prefetcher is limited by the branch predictor's accuracy and the BTB hit rate. Improving the FDIP further is an interesting area of research in terms of instruction prefetching.

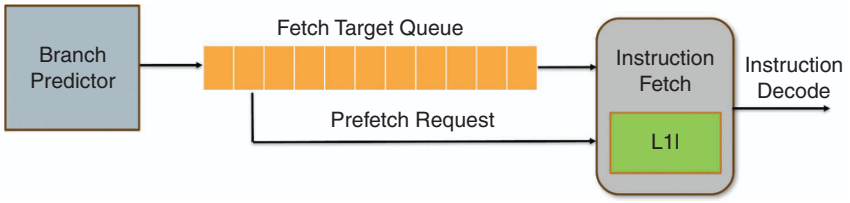

FIGURE 5. FDIP instruction prefetcher in action.

Prefetching in GPUs

So far in this article, we have discussed data prefetching in CPUs. However, interesting prefetching ideas can be applied to general-purpose GPUs (GPGPUs) too. Prefetching in GPGPUs is not that common as GPGPUs are bandwidth-sensitive computing engines where latency is not the primary concern. However, there are latency-sensitive GPU workloads, and hardware prefetching can be beneficial there. Software prefetching in GPUs is also shown to be effective, again for a specific set of workloads that are constrained by latency and not bandwidth.

Last but not least, can artificial intelligence (AI) help hardware prefetchers?

THE NEED FOR AI

The architecture community has extensively worked on understanding memory access patterns and proposed a variety of prefetchers. However, there are still many access patterns that are hard to predict and the usage of AI can help.[7,8] We are still in the early days of AI for hardware prefetching. The goal at this moment seems to answer the following question: What are the limits of hardware prefetching if we have all of the intelligence and no practical constraints like storage overhead and implementation complexity? The community is exploring AI and subfields of AI, like machine learning or reinforcement learning, for prefetching. Currently, the AI ideas for data prefetching are impractical, whereas reinforcement learning-based ones are practical, as most of them are extremely lightweight.[9] With the advent of new application domains, like graph analytics, recommendation systems, computer vision, etc., designing domain-specific hardware prefetchers is the key as these applications show different memory access patterns depending on the underlying data structures and the input provided, which makes them a combination of regular and irregular access patterns. An AI-based prefetcher can adapt and select the best based on the access patterns.

PREFETCHING AND SECURITY

Post the Spectre attacks,[10] various microarchitectural units are used for side and covert channels, and thus the hardware prefetcher is not an exception. Hardware prefetchers have been shown to mount cross-thread and cross-core covert channel attacks. Specific data prefetchers, like the data memory-dependent prefetchers, have created new side-channel attacks, like GoFetch.[11] These attacks pose a new set of challenges for the microarchitecture community as the design of high-performing, storage efficient, and secure prefetchers will be a need of the future. In general, security and performance provide conflicting tradeoffs as nothing comes for free in the world of microarchitecture optimizations.

In conclusion, the world of latency, bandwidth, and prefetching is here to stay with more interesting ideas and insights on the way. AI for prefetching, secure prefetching, and prefetching for new application domains are some of the interesting directions of future research in this exciting area.

ACKNOWLEDGMENT

Thanks to my graduate students Nishkarsh, Shubham, Vedant, and Naman, who helped me in shaping the article.

REFERENCES

1. Data Prefetch Support. Accessed: Jan. 2023. [Online]. Available: https://gcc.gnu.org/projects/prefetch.html
2. "The 3rd data prefetching championship." DPC3. Accessed: Jun. 2023. [Online]. Available: https://dpc3.compas.cs.stonybrook.edu/
3. "Hot Chips 2023: Arm's Neoverse V2." Chips and Cheese. Accessed: Sep. 2023. [Online]. Available: https://chipsandcheese.com/2023/09/11/hot-chips-2023-arms-neoverse-v2/
4. A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: An accurate local-delta data prefetcher," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 975–991, doi: 10.1109/MICRO56248.2022.00072.
5. A. Jamet, G. Vavouliotis, D. Jimenez, L. Alvarez, and M. Casas, "A two-level neural approach combining off-chip prediction with adaptive prefetch filtering," in *Proc. 30th IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2024, pp. 528–542.
6. B. Panda, "CLIP: Load Criticality based data prefetching for bandwidth-constrained many-core

systems," in *Proc. 56th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, pp. 714–727, doi: 10.1145/3613424.3614245.
7. Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A hierarchical neural model of data prefetching," in *Proc. 26th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2021, pp. 861–873, doi: 10.1145/3445814.3446752.
8. S. Mohapatra and B. Panda, "Drishyam: An image is worth a data prefetcher," in *Proc. 32nd Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2023, pp. 51–61, doi: 10.1109/PACT58117.2023.00013.
9. R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2021, pp. 1121–1137, doi: 10.1145/3466752.3480114.
10. P. Kocher et al., "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2019, pp. 1–19, doi: 10.1109/SP.2019.00002.
11. B. Chen et al., "GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers," in *Proc. USENIX Secur. Symp.*, 2024, pp. 1–21.

BISWABANDAN PANDA is with the Indian Institute of Technology Bombay Powai, Mumbai 400076, India. Contact him at biswa@cse.iitb.ac.in.

Digital Object Identifier 10.1109/MC.2024.3397371