Leveraging Artificial Intelligence Algorithms for Hardware Prediction

Research Plan

Department of Computer Architecture
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya

Author:
Georgios Vavouliotis
georgios.vavouliotis@bsc.es

Supervisors:
Marc Casas Guix
marc.casas@bsc.es
Lluc Alvarez Marti
lluc.alvarez@bsc.es

January 27, 2020

Contents

Chapter 1

Introduction

Moore's law and Dennard Scaling (or MOSFET Scaling) have driven nearly the last 40 years the electronics and semiconductor industries. Moore's law [75] states that roughly every 18 months, the number of transistors that can be squeezed onto an integrated circuit doubles at constant cost. From the economic point of view the shrunk in transistor area lowers the cost of transistor by 30% per year, i.e., the same chip can be bought for a lower price. In addition, Moore's law correlates well with single-thread performance because smaller transistors can switch at higher speeds. Dennard Scaling [36] postulated that as transistors get smaller their power density remains constant so that the power use stays in proportion with area. Taking advantage of Dennard Scaling the chip manufacturers have been producing chips that could operate in higher frequencies at the same power.

Nowadays, Moore's law is petering out because there are physical limits in the ability to continually shrink the size of components on a chip. The latest chips have billions of transistors that are invisible to the human eye. If Moore's law was to continue through the next 20 and more years, engineers would have to build transistors from components that are smaller than a single atom of hydrogen. Moreover, it is also increasingly expensive for companies to keep up because building fabrication plants for new chips costs billions.

Dennard Scaling does not take into account the leakage current and the threshold voltage. Specifically, as we moving to smaller technologies (e.g. 7nm, 5nm) the channel that carries the electrical current through the transistor cannot always contain it. This generates heat which can wear out the transistors more quickly, making them even more susceptible to leakage. Heat is not just limited to one transistor though. Billions of transistors leaking can seriously threaten the integrity of the whole chip, so the processor must reduce the amount of voltage it takes in or throttle the number of transistors in use to prevent overheating, limiting the processing power of the chip. The inability to operate within the same power envelope for these technologies led to Dennard Scaling breakdown at 2006. This has limited practical processor frequency to around 4 GHz and is commonly referred as the Power Wall.

The slowing of Moore's law and the end of Dennard Scaling does not mean that the progress will come to a complete stop. The law-customer expectations continue to grow rapidly, hence, innovative techniques that

exploit different levels of abstraction should be examined and promoted in order to replace the conventional approaches. For example, the stagnated processor clock frequencies encouraged engineers and researchers to shift to multi-core designs [58,66] and eventually nowadays the majority of the modern computing systems are multi-core while there is a lot of in-flight research on this field. However, as the different cores compete for the shared system resources they can saturate them. This contention can be caused due to memory bandwidth saturation if the service of multiple cores is needed or higher miss rates (memory latency increase) if different cores use the same memory space. This is known as the Memory Wall [72, 79, 96].

Even if Moore's law was to end tomorrow, there are plenty of pioneering ideas that could help progress to move forward. From the hardware side, the design of more specialized chips like Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) for applications that heavily stress the general purpose chips boast a range of improvements, such as greater levels of performance per watt. In addition, heterogeneous cores and memory systems are gaining more and more attention because they face the Memory Wall problem. Approximate computing is also a famous technique to reduce the great overheads of modern applications with enormous needs in terms of computations and data movements. From the software side, there is plenty of room for squeezing more performance out of the same chips by optimizing today's software and improving the algorithms that are being used. Finally, there is a lot of pending research on building chips using materials other than silicon (e.g. gallium nitride).

Artificial Intelligence (AI) and Machine Learning (ML) are the two most frequently used terms in computer science and often seem to be used interchangeably. To differentiate between them, Artificial Intelligence can be considered as the broader concept of machines being able to carry out tasks in a similar way that a person would do and Machine Learning can be considered as an application of AI based around the idea that we should just be able to give machines access to data and let them learn for themselves. Deep Neural Networks (DNNs), computer vision applications like self-driving cars, voice recognition and sentimental analysis have huge requirements in terms of computational resources and energy consumption. All the modern general purpose computing systems suffer from the significant performance downward and the energy consumption increase that these applications cause. The solutions that have been proposed to partially address this kind of applications add more complexity in the already complex design of the newest multi-core systems. As a result, at the hardware level there is a large amount of available information that non of the existing systems fully exploit. These information could be fed in a sophisticated way into intelligent hardware components that could take advantage of them and alleviate the system by improving performance and reducing the energy consumption.

To summarize, a new golden age for computer architecture is rising as non conventional and groundbreaking approaches are needed to overcome the barrier that is placed by end of Dennard Scaling and the slowing of Moore's law.

Chapter 2

Motivation

In this section we explain the reasons why it is promising to integrate Artificial Intelligence (AI) techniques into novel computer architecture components from the performance and energy consumption perspectives. In addition, we present specific hardware and software areas that need innovative approaches and we explain why AI-based techniques are compatible with the requirements of these domains.

2.1 Interaction between Artificial Intelligence and Computer Architecture

The multiple challenges that current multi-core architectures face in terms of energy consumption, data movement and massive concurrency are making them increasingly complex. Indeed, very sophisticated memory systems and on-chip interconnects are being designed to mitigate data movement bottlenecks. To make the current situation even more complicated, the scope of application domains targeted by general-purpose computer architectures goes beyond the traditional computationally intensive workloads or data compression algorithms and includes emerging areas like social networks, autonomous driving and data analytics. A key aspect brought by the increasing complexity of current multi-core designs is the large amount of information available at the hardware and the runtime system levels, which neither current architectures nor software stacks fully exploit despite its potential for enabling very significant improvements in terms of performance and energy consumption.

In the recent literature there are tons of proposed AI techniques aimed at solving a variety of issues; from simple classification problems up to self-driving cars and sentimental analysis. Some of these AI learning techniques (e.g. rule-based reasoning, multi-layer perceptrons, decision trees, support vector machines and neural networks) could be modified and optimized to be efficiently integrated onto novel computer architecture components. The main reasons why make sense to do research on building 'smart' hardware components that include AI-based techniques are: (i) different applications have different memory access patterns and type of available information which can be detected and accordingly served by a 'smart' hardware component, and (ii) all the AI techniques are highly accurate and precise, avoiding useless data movements and computations, and hence contribute in the reduction of energy consumption and data movements. None of the conventional

approaches is capable of satisfying one of the above mentioned objectives. Contrarily, the 'smart' hardware components could fully exploit the available information on the different computer architecture layers and based on their learning algorithm they could provide benefits such as precise prediction capabilities and trigger countermeasures when required.

2.2 Integrating Intelligence into Hardware

In this section we briefly present specific domains where intelligent hardware components could make a big difference in terms of performance and energy consumption.

Hardware/Software Co-designed Prediction Mechanisms. The building blocks of this novel approach will be: (i) precise and extremely lightweight prediction capabilities based on machine learning approaches (e.g. multi-layer perceptron) applied at the hardware and the system software levels, and (ii) hardware-software mechanisms able to trigger countermeasures to immediately take action and exploit AI-based predictions. For example, hardware-based prediction mechanisms will be able to anticipate suboptimal execution regimes like execution phases dominated by cache misses and trigger countermeasures like informed prefetching policies. To do so, this approach is aimed at designing new hardware components with novel prediction and reconfiguration capabilities as well as using existing hardware in a more efficient way.

Micro-Architecture Predictors. This approach targets the design of micro-architecture predictors based on machine learning techniques (e.g. perceptron learning) to support hardware predictions and decisions. For example, in a context with two data prefetchers, one of them cheap and state-of-the-art but likely to bring useless data and the other an expensive multi-layer perceptron prefetcher but capable of bringing the appropriate cache lines, this approach will produce fast and machine learning-based hardware methods to predict when worth using the cheap prefetcher instead of the expensive one. Another example is DNN-based inference in branch target prediction which involves anticipating the branch target besides whether or not the branch is taken or not. This kind of learning-driven support to hardware decisions will significantly improve the way hardware use its own resources and avoid wasting power and CPU cycles.

Cache Management. Aspects like cache insertion and promotion are largely studied in the literature. However, there is still much room for improvement since current methods are far from reaching the most optimal policies per each application. Approaches such as analyzing the impact of training and using Deep Neural Networks (DNNs) to guide cache management decisions like cache insertion and promotion policies have never been used due to their implementation costs. Decreasing the feature size or the latency involved in obtaining the DNN's inferences are ways to overcome the implementation cost of this approach. The latency issue will be worked out by generating the inference during idle cycles due to cache misses or page faults. The goal is to promote those cache lines identified as important by a DNN-based inference process every time that an event like a cache miss or a page faults provides some room for it.

Other Research Areas. Besides the areas described in the previous sections, AI-based techniques can be leveraged to other relevant challenges like guiding memory allocation decisions across large systems, which involve the movement of memory pages to increase locality, or driving scheduling and reconfiguring decisions on heterogeneous systems composed of different kinds of computing devices. In general, any scenario where large amounts of data are available at the hardware or the system software levels is suitable for integrating AI approaches into hardware.

Chapter 3

Related Work

After Dennard Scaling breakdown and the slowing of Moore's law the vast majority of the computer architects shifted to multi-core system design, facing new challenges and different kind of tradeoffs that require much more complex designs to be solved. Data movement, shared resources among different cores or multiple threads are the main reasons for performance degradation in such systems. Especially for applications with huge data and code footprints the performance bottleneck is even bigger and it has been observed huge increase in the energy consumption when this type of applications run in a system. As explained in Section 2.1, Artificial Intelligence (AI) algorithms are very suitable for hardware prediction schemes and cache management policies, thus, they could be integrated into hardware components to alleviate system's performance and reduce the energy consumption. In this section we analyze already proposed hardware mechanisms based on AI techniques and we elaborate on the key idea behind their design in order to highlight the benefits that AI brought in the different parts of computer architecture. The majority of these mechanisms have been proposed by academia and some of them have been included into fabricated chips.

3.1 Branch Prediction

Branch prediction is the first area that an AI-based approach appeared in the computer architecture research field. Jiménez *et al.* [54] proposed a new method for dynamic branch prediction that includes the simplest type of neural network, the perceptron, as an alternative to the commonly used two-bit saturating counters. The perceptron-based branch predictor managed to improve the misprediction rate over the state-of-the-art predictor by 10%. One of the key aspects of this groundbreaking predictor is that the hardware resources scale linearly with the history length. At this time, other purely dynamic branch prediction schemes required exponential resources.

The publication of the dynamic branch predictor with perceptrons [54] triggered a lot of research on this area using AI-based techniques. As a result, many innovative branch prediction approaches have been proposed since then [48–51, 55, 56, 90]. The vast majority of them continue to use perceptron as building block of their neural branch predictor because a perceptron-based scheme requires low additional hardware

complexity and it is able to produce the corresponding output within 1-2 cpu cycles. The Piecewise Linear Branch Prediction [50] includes a set of linear functions, one for each program path to the branch to be predicted, that separate predicted taken from predicted not taken branches. This branch predictor unifies the previously distinct concepts of perceptron prediction [54] and path-based neural prediction [48] and manages to significantly improve accuracy over the already proposed predictors while adding modest hardware complexity.

Recently, Samira *et al.* [74] proposed a replacement technique that uses the history of past instruction addresses and their reuse behaviors to predict dead blocks in the instruction cache (I-cache) and dead entries in the branch target buffer (BTB), i.e. entries that they would not be reused before they are evicted from the cache. Specifically, the proposed replacement policy is enhanced with the bypass optimization. This work reveals that perceptron-based predictive replacement policies significantly reduce misses in the I-cache and BTB compared to the already proposed approaches.

3.2 Cache Management

Regarding the cache management research domain, all the novel works that are based on an AI approach are targeting to alleviate the last level cache (LLC) performance because this cache can be quite inefficient and has great power requirements due to its size (2-8 MB). For the rest of this section we are going to refer to dead entries and dead blocks. An entry or block is considered dead when is not gonna be reused before its eviction from the cache. Samira *et al.* [65] introduced the sampling dead block predictor, a scheme that samples program counters (PCs) to determine when a cache block is likely to be dead. Rather than learning from accesses in the cache sets, the sampling predictor keeps track of a small number of sets using partial tags. Sampling allows the predictor to use far less state than previous predictors to make predictions with superior accuracy. Jiménez [52] proposed the space-efficient Tree-based PseudoLRU replacement policy which is based on representing block positions as distinct paths in a binary tree. Elvira *et al.* [93] found that a placement or promotion for one block often needlessly disturbs the non-promoted blocks and hence guided by the principle of minimal disturbance, i.e. that a policy should seek to disturb the order of non-promoted blocks to the smallest extent possible, developed a simple modification to PseudoLRU resulting in a policy that improves performance over previous techniques while retaining the low cost of PseudoLRU. Moreover, Elvira *et al.* [94] motivated by the disparity between last-level cache (LLC) and memory latencies proposed the perceptron learning for reuse prediction, a pure AI-based replacement policy for the LLC enhanced with the bypass optimization. This predictor managed to significantly improve the accuracy over the state-of-the-art reuse predictor due to the perceptron learning. Jiménez *et al.* [57] proposed a more sophisticated dead block predictor that uses a larger set of features as input to the perceptron-based learning algorithm. The bigger set of features gives the opportunity to detect more correlations between access patterns that conclude to better decisions and finally efficiency and accuracy. Moreover, this predictor also includes placement, promotion, and bypass optimizations inspired by previous work [93].

3.3 Hardware Prefetching Schemes

The idea of doing prefetching was originally introduced to reduce the performance gap between processor, caches and main memory. Cavazos *et al.* [33] states that doing prefetching for applications with completely

different and complicated access patterns is amenable to neural networks. Since then, a lot of research has been done trying to efficiently merge hardware prefetching with sophisticated AI techniques. Liao *et al.* [70] develops a tuning framework for data center applications which predicts the optimal prefetch configuration based on hardware performance counters. Hashemi *et al.* [43] demonstrates the potential of deep learning in the critical problem of learning memory access patterns. This work relates contemporary prefetching approaches to n-gram models in natural language processing and shows that recurrent neural networks yield superior precision and recall compared to the state-of-the-art prefetchers. Peled *et al.* [80] introduced the context-based memory prefetcher which approximates the semantic locality concept using reinforcement learning. Finally, a long short term memory (LSTM) hardware prefether has been recently proposed by [98]. This AI-based prefetching scheme is aimed at detecting complex memory access patterns in multicore systems. The experimental results show that this LSTM prefetcher achieve higher accuracy and better coverage on the evaluated set of workloads.

3.4 Memory Controllers

Ipek *et al.* [45] proposed a self-optimizing memory controller, an idea which is AI-based since it includes reinforcement learning. Conventional memory controllers deliver relatively low performance in part because they often employ fixed, rigid access scheduling policies designed for average-case application behavior. As a result, they cannot learn and optimize the long-term performance impact of their scheduling decisions, and cannot adapt their scheduling policies to dynamic workload behavior. The proposed memory controller overcomes these limitations by using the principles of reinforcement learning.

Chapter 4

Research Plan

In this section we discuss the main objectives of this PhD program and we present the plan of our research work. We highlight the main milestones of the program and we elaborate on the way we are going to reach them. Figure 4.1 presents a Gantt diagram of how we are going to distribute the different research targets through the three year PhD program. Black stripes indicate the already completed stages, while the red colored ones denote the pending stages. The following sections explain in detail the different objectives presented in the Gantt diagram.

4.1 Literature Review

During the first months of the PhD program we reviewed publications about how modern computing systems work and their main bottlenecks. Focusing on the computer architecture part, we identified possible room for improvement in the following fields: (i) virtual memory management, (ii) scale-out applications and their implications in the memory hierarchy, and (iii) branch prediction; wrong path execution and hard-to-predict branches.

The virtual memory management approach focuses on the performance bottleneck of the Translation Lookaside Buffer (TLB) and is aimed at saving future TLB misses by prefetching page table entries before they are accessed. The second approach entails a big set of scale-out applications with huge code and memory footprints [23, 38, 60, 67] that possibly stress components of a memory hierarchy like first level instruction cache (L1I), second level unified cache (L2) and instruction TLB. The goal is to design predictors and policies driven by AI algorithms that minimize the bottlenecks caused by scale-out applications and also have the required properties to be efficiently integrated into modern hardware designs. The final research domain of this PhD program deals with branch prediction. Since the latest branch predictors have almost 98% accuracy is hard to design a novel branch predictor that significantly outperforms the state-of-the-art. However, there is room for improvement in the fields of wrong path execution and hard-to-predict branches. We intend to revisit already proposed techniques, extend them and enhance them with AI-based algorithms capable of detecting phases that the program is in the wrong path and recover it and also identify hard-to-predict

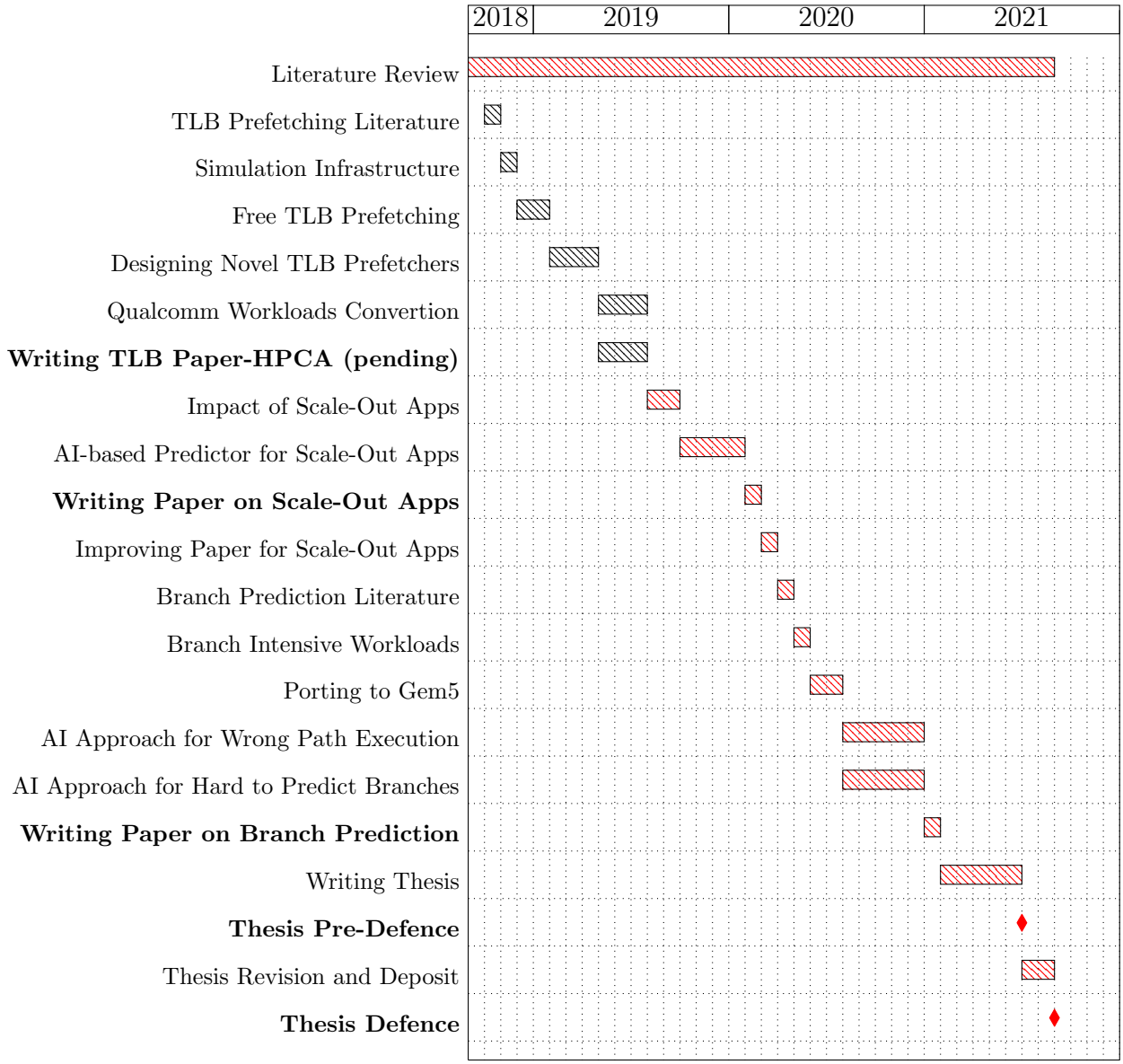

Figure 4.1: Research plan presented in a Gantt diagram.

branches before letting the predictor decide about the branch outcome. We elaborate on the requirements of the above briefly described research approaches in the following sections.

Finally, reviewing the latest literature on the research domains that we work on is an ongoing process through the whole PhD program. We plan to stay on top of the latest on our research fields so our work remains highly innovative and competitive.

4.2 Cost-Effective TLB Prefetching

This work is aimed at alleviating system from the TLB performance bottleneck. Prior work has quantified the cost of TLB performance [21,28,30,63,85] and proposed approaches to mitigate the performance overhead of address translation. These approaches mainly fall into three categories: (i) increasing the TLB reach by introducing hardware and OS support for enhancing and exploiting contiguity [19,28,34,40,42,61,62,71,76–78,82–84,89,91,92], (ii) reducing the latency of TLB misses [20,26,27,29,30,87,97], and (iii) eliminating TLB misses by prefetching page table entries [24,32,59,88,95]. In this work we target this last category that is independent of the state of the system and depends only on the memory access pattern of the application.

In prior work on TLB prefetching, prefetch mechanisms always trigger a page walk in the background to prefetch a page table entry (PTE) [32,59,88]. Thus, TLB prefetching is considered costly in terms of memory operations due to the large number of page walks required. However, the cost of TLB prefetching can be significantly reduced thanks to the locality of PTEs in the last level of the page table [31,82,83].

The first contribution of our work is the introduction of *free prefetching* for TLBs, a technique that exploits the locality in the page table to prefetch multiple PTEs within a single page walk. Free prefetching identifies and fetches contiguous PTEs that are transferred through the cache hierarchy into a TLB prefetch queue at the end of a page walk. In addition, free prefetching is transparent among different TLB prefetching schemes, it does not require extra hardware to be implemented, and it permits aggressive TLB prefetching because it is not affected by limited page table walker ports. Prior work [31,82,83] also leverages this locality to increase TLB reach and reduce page walks, but none of them exploits it from the perspective of prefetching PTEs for per-core private TLBs. The introduction of free prefetching into state-of-the-art TLB prefetchers and our proposed TLB prefetching schemes provides great performance improvements while eliminating the vast majority of required page walks.

To elaborate on PTE locality, consider a system that uses the x86-64 architecture. Note that we focus on virtual memory used in x86-64 architectures [1,11] which is fundamentally similar to other architectures [2,18]. The page table is organized as a four-level radix tree [10] with levels named PML4, PDP, PD, and PT from root to leaves. Some vendors support a five-level radix tree [9]. Figure 4.2 explains the procedure that takes place on a page walk and exposes the locality of the PTEs in the last level of the page table. The virtual address is split into groups of bits that provide the corresponding offset for each page table level. The first level (PML4) is indexed using the CR3 register. Using the PML4 offset, the page table walker finds the entry of PML4 level that points to the next level. Following the same procedure for the PDP and PD levels, the page walker finds the virtual address of the PTE that stores the requested translation. Finally, a memory access is issued to fetch the PTE which is placed in a cache line to be sent to the TLB through the cache hierarchy.

In x86-64 architectures a PTE is 8 bytes and a cache line is 64 bytes. Thus, a single cache line can store 8 PTEs. When the requested PTE is read from memory at the end of a page walk, the chance for free prefetching arises as the requested PTE is grouped with the 7 neighboring PTEs and forming together a single cache line. That cache line will hold the requested translation plus 7 more translations that do not require any additional memory operations to be fetched. Hence, free prefetching exploits these opportunities to prefetch multiple consecutive PTES within a single page walk into a TLB prefetch queue.

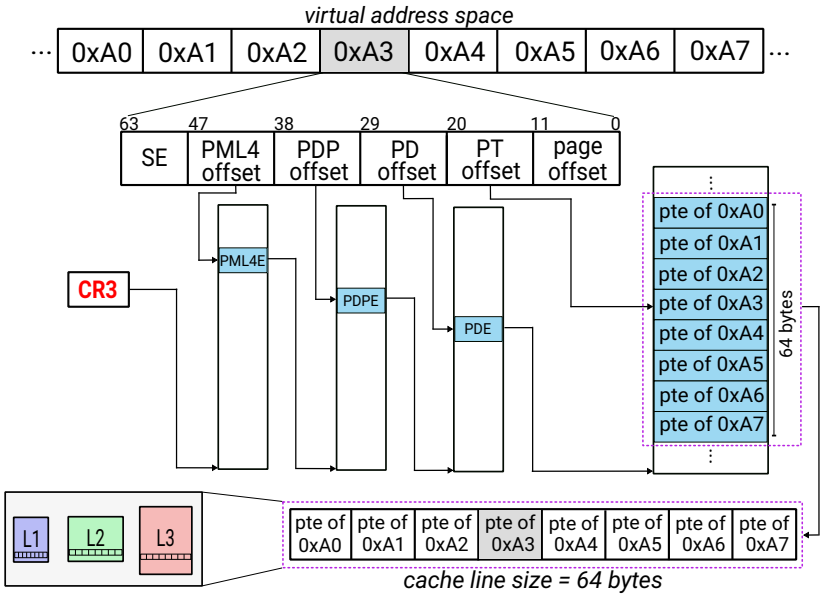

Figure 4.2: Locality of the PTEs in the last level of the page table, assuming a page walk for a random virtual page.

In addition, motivated by the pressure on the TLB by workloads with huge code and memory footprints [23, 38, 60, 67], we design a set of cost-effective TLB prefetching schemes to overcome the TLB performance bottleneck. We propose a modified and more aggressive version of Arbitrary Stride Prefetcher (ASP) [24, 59], a previously proposed TLB prefetcher. Moreover, we propose novel low-complexity TLB prefetchers that significantly outperform the state-of-the-art. Our analysis indicates that no single TLB prefetcher performs best among all the applications and some workloads do not benefit from TLB prefetching due to irregular patterns. Driven by these findings, we propose hybrid and tournament TLB prefetching schemes that efficiently combine multiple prefetchers to obtain high performance. The hybrid scheme minimizes hardware complexity at the expense of triggering more page walks while the tournament scheme reduces page walks at the expense of needing additional hardware. The components of the hybrid scheme operate individually and the tournament scheme is enhanced with two mechanisms: a selection logic that dynamically enables the most accurate prefetcher and a throttling mechanism that eliminates page walks by discarding useless prefetches. Next, we briefly present the main contributions of this work:

- We revisit the TLB prefetching concept and we evaluate the state-of-the-art TLB prefetchers using a set of 2000 modern industrial and academic workloads.
- We introduce free prefetching for TLBs, a mechanism that identifies and fetches multiple page table entries that are transferred through the cache hierarchy at the end of each page walk. Free prefetching improves geometric mean speedup up to 5.4%, eliminates up to 96% of required page walks for prefetching, and reduces on average the total page walks up to 37% compared to no TLB prefetching.
- We propose MASP, an aggressive evolution of a state-of-the-art TLB prefetcher ASP [24, 59]. MASP improves geometric mean speedup up to 6.0% compared to ASP.
- We propose HiSH, a hybrid TLB prefetcher that minimizes hardware complexity at the expense of triggering more page walks. HiSH achieves geometric mean speedups up to 13.3% and 8.9% for the

academic and industrial workloads over no TLB prefetching, respectively with only 0.68KB additional hardware complexity.

- We propose T^{3}P, an AI-based tournament TLB prefetching scheme. T^{3}P is implemented as a decision tree, a machine learning algorithm for supervised learning explained in Section 5.3. T^{3}P is enhanced with two mechanisms: a selection logic that dynamically enables the most accurate prefetcher; and a throttling mechanism that eliminates page walks by dynamically discarding useless prefetches. T^{3}P yields geometric mean speedup up to 15% and 9.7% with 18% and 32% average reduction on page walks for the academic and industrial workloads over no TLB prefetching, respectively.

The simulation infrastructure of this work is based on ChampSim [5], a trace based simulator that measures the performance impact of TLB prefetching. More information about ChampSim exist in Section 5.1. The set of workloads consists of 2000 industrial traces of high performance computing and server applications provided by Qualcomm for CVP1 contest [4] and all the benchmarks from SPEC CPU 2006 [16, 44] and SPEC CPU 2017 [17]. Workloads with a TLB MPKI of at least 1.0 are considered TLB intensive and thus taken into account in our experimental campaign. More information about the set workloads exist in Section 5.2.

4.3 Scale-Out Applications

Our research approach on the domain of scale-out applications has the following requirements: (1) collect a large set of scale-out applications, (2) select the appropriate simulation infrastructure based on the format of the applications, (3) find out the main bottlenecks when scale-out applications are executed, and (4) work on new techniques and finally propose designs that are able to eliminate the negative effects of these applications.

Simulation Infrastructure. The simulation infrastructure that we will use for this research approach is ChampSim [5], a trace-based simulator. An extended presentation of ChampSim's characteristics is presented in Section 5.1. The main reason why we selected ChampSim is because it is difficult to find the source code of industry-sourced scale-out applications.

Collecting Scale-Out Applications. The traces that were used for the first Championship on Value Prediction (CVP1) [4] contain a set of over 2000 industrial traces of high performance computing (HPC) and server applications provided by Qualcomm. These traces were in a format that is readable only by the contest's simulator which source code was not publicly available. Hence, to port all these traces in ChampSim we had to convert them into ChampSim readable. Currently, all these traces are successfully converted and ported to ChampSim. Note that for this research approach we only consider the server applications from the Qualcomm workloads. Finally, the set of workloads could be extended in the future if we find scale-out applications that can be properly traced and ported to ChampSim.

Work Plan. As first step, we plan to examine the behavior of the scale-out applications using metrics like Misses per Kilo Instructions (MPKI). Based on these results we will find out how much these applications stress the different architectural components (e.g. first level instruction cache, second level unified cache and instruction TLB). Based on the experimental results we will conclude if there is room for improvement

in this field. Preliminary results look promising, however there is a large set of pending simulations and many unexplored scenarios. If the analysis stage detects significant opportunities for improvements, we plan to implement, evaluate and propose an AI-based instruction prefetcher for scale-out applications aimed at alleviating system's performance and reducing the energy consumption by preloading instructions in the first level instruction cache or instruction TLB. The main AI algorithms that we get inspired of for the design of this intelligent instruction prefetcher are presented in Section 5.3.

4.4 Branch Prediction

The final research domain that we plan to work on is branch prediction, one of the most famous research areas in computer architecture. The branch prediction accuracy has been stagnated for the last years. However, the accuracy of a modern branch predictor is close to 98%. This PhD program targets to deal with two major problems on the branch prediction area: the hard-to-predict branches and the consequences of the wrong path execution.

Hard to predict branches are branches that a modern and highly accurate branch predictor is failing most of time. These branches are the main reason for the branch prediction accuracy stagnation. The term wrong path execution refers to the case that a branch predictor fails to predict the outcome of branch and then continues to execute instructions that will be flushed from the pipeline when the branch will be resolved. Moreover, wrong path execution pollutes the branch prediction structures. It has been shown that taking the wrong path concludes to significant performance degradation [22, 37, 41, 46, 53]. Hence, mechanisms that are able to mitigate the overhead of wrong path execution are important for modern computing systems. We plan to design smart AI-based mechanisms that are able to detect cases that the wrong path is taken and force the execution to recover from it. Examining the wrong path execution and understanding its effects requires a cycle-accurate simulator. Thus, in this research domain we plan to use the Gem5 simulator. A complete description of this simulator and its capabilities is presented in Section 5.1. The main AI algorithms that we will use on the branch prediction research domain are presented in Section 5.3.

We have not yet intensively dealt with the branch prediction research area, thus, we do not have any experimental results to present at this point.

Chapter 5

Methodology

The main targets of this PhD program are: (i) detect cases where the conventional hardware is not capable of maintaining the desirable performance and energy consumption, (ii) for theses cases explore different AI-based approaches and optimize them with respect to the hardware design principles, and (iii) evaluate these designs and provide meaningful conclusions to the scientific community. Every single target of this PhD program requires: (i) a globally acceptable simulation environment capable of precisely simulating the behavior of a modern computing system and (ii) a set of modern academic and industry-sourced workloads to evaluate the proposed designs. To fulfill these requirements, we use two well-known and highly accurate simulators to measure the performance gains of our designs compared to the state-of-the-art approaches, Gem5 [8] and ChampSim [5]. We present the simulators and the set of workloads that we consider in Sections 5.1 and 5.2, respectively.

5.1 Simulation Infrastructures

In this section we describe the different simulation infrastructures that we use to evaluate our work.

Gem5. Gem5 [8] is a event-accurate full system simulator that has been used for many years by the computer architecture community and its results are globally accepted. Gem5 is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture. The developers of Gem5 have already simulated every component that exists and every event that happens in a modern computing system.

ChampSim. The second simulator that will be used in this PhD program, ChampSim, is a trace-driven simulator which is useful for microarchitecture studies. ChampSim has been successfully used by the 2nd Cache Replacement Championship (CRC-2) [5] and the 3rd Data Prefetching Championship (DPC-3) [7]. In addition, significant work using ChampSim has been published into accredited computer architecture conferences [25, 47]. ChampSim simulates a modern out-of-order CPU with realistic sized microarchitectural

structures, has a robust cache and DRAM models and is open-source, thus everyone can download it, use it and extend it.

Energy and Hardware Complexity Models. Power budget and hardware complexity are the most critical constraints that have to be taken into consideration in the design of a modern multi-core system. To complement our performance results with energy, hardware complexity and on chip area requirements measurements we use: (i) Cacti 6.5 [69] with the latest technology available and (ii) McPAT [68], an integrated power, area, and timing modeling framework for multi-core architectures.

5.2 Workloads

Modern workloads that congest significant bottlenecks in terms of performance and energy consumption in the most recent computing systems are necessary for the scope of this PhD program. For this reason, all the proposed designs will be evaluated using a big set of both academic and industry-sources workloads. As explained in Section 5.1 we use two different simulators, Gem5 and ChampSim, which are event-accurate and trace driven, respectively. Thus, each simulator has different needs in terms of workload formant. We elaborate on this issue in the following sections. Finally, the complete set of workloads that we consider in this PhD program to assess our proposed designs are presented in Table 5.1.

Gem5 workloads. Gem5 requires only the source code of a workload and the corresponding input sets in order to be able to simulate its execution. Specifically, we use all the workloads included in the well-known SPEC CPU 2006 [16, 44] and SPEC CPU 2017 [17] benchmark suites.

ChampSim workloads. Since ChampSim is trace-based simulator requires traces of applications to perform. To satisfy the need for traces, we use the tracing mechanism that is already included in the current version of ChampSim and the dynamic binary instrumentation tool Pin [14]. After the trace conversion, ChampSim is capable of simulating the execution of the corresponding workload. All the workloads included in the SPEC CPU 2006 [16, 44] and SPEC CPU 2017 [17] benchmark suites have been already converted into ChampSim readable traces. For the SPEC workloads we also use SimPoint [81] to identify up to 6 weighted simpoints of different program phases per benchmark and Pin [14] to produce one trace per simpoint. For the experimental results we report the weighted average of simpoints for each SPEC workload. In order to include industry-sourced workloads in our research, we decided to use a set of over 2000 industrial traces of high performance computing and server applications provided by Qualcomm for first Championship on Value Prediction (CVP1) [4]. Despite the fact that these traces were not in a format that is readable from ChampSim we managed to convert them into ChampSim readable ones and include them in our set of workloads.

5.3 Arificial Intelligence Algorithms

In this section we present the main AI algorithms that we plan to use in the design of the proposed hardware prediction schemes.

SPEC CPU 2006	perlbenc, bzip2, gcc, bwaves, games, mcf, milc, zeusmp, gromacs, cactusADM, leslie3D, namd, gobmk, dealII, soplex, povray, calculix, hmmer, sjeng, GemsFDTD, libquantum, h264ref, tonto, lbm, omentpp, astar, wrf, sphinx3, xalancbmk
SPEC CPU 2017	perlbench_s, gcc_s, bwaves_s, mcf_s, cactuBSSN_s, lbm_s, omnetpp_s, wrf_s, xalancbmk_s, x264_s, cam4_s, pop2_s, deepsjeng_s, imagick_s, leela_s, nab_s, exchange2_s, fotonik3d_s, roms_s, xz_s
Qualcomm	400 Floating Point HPC Traces, 800 Integer HPC Traces, 800 Server Traces and 200 Crypto Traces

Table 5.1: Complete set of considered workloads.

Decision Trees. A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility [6]. A decision tree is a flowchart-like structure in which each internal node represents a 'test' on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. Tree based learning algorithms are considered to be one of the best and mostly used supervised methods for machine learning. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of classification or regression problem at hand. In the literature Decision Tree algorithms are referred also as Classification and Regression Trees (CART).

Significant works published in accredited conferences have already integrated decision trees into sophisticated replacement policies for the last level cache [52, 64, 93]. These works exploit the supervised learning benefits of the decision trees in order to determine the optimal insertion policy. More information about these works are presented in Section 3.2.

Perceptrons. In machine learning, a perceptron is an algorithm for supervised learning of binary classifiers [13, 73]. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class [39]. Perceptron is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. Specifically, the perceptron is an algorithm for learning a binary classifier called a threshold function: a function that maps its input X (a real-valued vector) to an output value $f(X)$ (a single binary value):

$$f(X) = \begin{cases} 1 & \text{if } w * X + b > 0, \\ 0 & otherwise \end{cases} \tag{5.1}$$

In the definition of function $f(X)$, w is a vector of real-valued weights, and w * X represents the dot product $\sum_{n=1}^{m} x_i w_i$, where m is the number of inputs to the perceptron, and b is the bias. The bias shifts the decision boundary away from the origin and does not depend on any input value. The value $f(X)$ is used to classify X as either a positive or a negative instance, in the case of a binary classification problem. If b is negative, then the weighted combination of inputs must produce a positive value greater than $|b|$ in order to push the classifier neuron over the 0 threshold. Spatially, the bias alters the position (though not the orientation) of the decision boundary. The perceptron learning algorithm does not terminate if the learning

set is not linearly separable. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly.

In the context of neural networks, a perceptron is an artificial neuron using the Heaviside step function as the activation function. The perceptron algorithm is also termed the single-layer perceptron, to distinguish it from a multilayer perceptron, which is a misnomer for a much more complicated neural network. As a linear classifier, the single-layer perceptron is the simplest feedforward neural network.

Neural Networks. Artificial neural networks (ANN) are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains [3]. Such systems 'learn' to perform tasks by considering examples, generally without being programmed with task-specific rules. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. ANNs have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and medical diagnosis.

As explained above the perceptron is the simplest feedforward artificial neural network. This PhD program is also aimed at using more complicated neural networks to build sophisticated hardware predictors and replacement policies. For example, the multi-layer perceptron (MLP) [12] is a class of complicated feedforward artificial neural network that consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a non-linear activation function. MLP utilizes a supervised learning technique called backpropagation [86] for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. MLP is able to distinguish data that is not linearly separable, a significant property that linear perceptron does not have [35].

We also plan to explore the area of Recurrent Neural Network (RNNs) [15], a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. Long short-term memory networks (LSTMs) are a concrete example of a RNN. Recent works [43, 98] used LSTMs for hardware prediction and highlighted their advantages over the conventional approaches.

The idea of integrating neural networks in the hardware design is an unexplored research area due to the constraints applied from the hardware in terms of complexity and energy consumption. We plan to overcome these limitations by simplifying the design of these neural networks (e.g. decrease number of features) on the expense of loosing some accuracy and generating the inference when costly events occur (e.g. page faults).

Chapter 6

Conclusions

The enormous bottlenecks that multi-core architectures face in terms of energy consumption and data movements are making the architectural design increasingly complex. Hence, there is a large amount of information available at the different hardware layers that conventional components are not able to exploit, missing then a lot of opportunities to increase the performance of the system and reduce the energy consumption. This PhD program is aimed at detecting and understanding problems in the hardware of modern computing systems and design AI-based prediction mechanisms to mitigate them.

In the case of systems that use virtual memory we revisited the concept of doing prefetching for the TLB. We evaluated the state-of-the-art TLB prefetching schemes and we observed that are not capable of detecting and predicting the future memory accesses for modern academic and industrial workloads. We proposed the free prefetching technique for per-core private TLBs, a technique that takes advantage of the locality in the page table to prefetch multiple consecutive page table entries within a single page walk. Moreover, we designed novel single, hybrid and tournament TLB prefetching schemes that exploit the free prefetching technique. The experimental results show that the proposed TLB prefetching schemes significantly outperform the state-of-the-art TLB prefetchers for all the evaluated workloads. Moreover, it has been observed huge reduction on the number of required page table walks when free prefetching is enabled over no TLB prefetching. The reduction of page walks degrades the energy spent for address translation. The works on scale-out applications and branch prediction are not yet finished, but we are currently working on them.

The research plan that we follow aims at designing novel hardware prediction schemes based on AI techniques and make significant contributions to the computer architecture community. Each contribution of our work will lead to a publication to a accredited international conference or journal in the field of computer architecture. At this point, we have under review one paper in the International Symposium on High-Performance Computer Architecture (HPCA). The rest of the works will be published in the most prestigious publishers in the computer architecture field.

Bibliography

[1] AMD64 Architecture Programmer Manual(Volume 2). https://www.amd.com/system/files/TechDocs/24593.pdf. [Online].

[2] ARMv8 Architecture Reference Manual. ARM. https://static.docs.arm.com/ddi0553/a/DDI0553A_e_armv8m_arm.pdf. [Online].

[3] Artificial Neural Network. https://en.wikipedia.org/wiki/Artificial_neural_network/. [Online].

[4] Championship Value Prediction (CVP). https://www.microarch.org/cvp1/. [Online].

[5] ChampSim Simulator. https://crc2.ece.tamu.edu/. [Online].

[6] Decision Trees. https://en.wikipedia.org/wiki/Decision_tree/. [Online].

[7] DPC3. https://dpc3.compas.cs.stonybrook.edu/. [Online].

[8] Gem5 Simulator. http://gem5.org/Main_Page/. [Online].

[9] Intel.5-Level Paging and 5-Level EPT. https://software.intel.com/sites/default/files/managed/2b/80/5-level_paging_white_paper.pdf. [Online].

[10] Intel® 64 and IA-32 Architectures Optimization Reference Manual. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf. [Online].

[11] Intel® 64 and IA-32 Architectures Software Developer Manuals. https://software.intel.com/en-us/articles/intel-sdm. [Online].

[12] Multilayer Perceptron. https://en.wikipedia.org/wiki/Multilayer_perceptron/. [Online].

[13] Perceptron. https://en.wikipedia.org/wiki/Perceptron/. [Online].

[14] Pin - A Dynamic Binary Instrumentation Tool. https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool. [Online].

[15] Recurrent Neural Network. https://en.wikipedia.org/wiki/Recurrent_neural_network/. [Online].

[16] SPEC CPU 2006. https://www.spec.org/cpu2006/. [Online].

[17] SPEC CPU 2017. https://www.spec.org/cpu2017/. [Online].

[18] The RISC-V Instruction Set Manual. https://content.riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf. [Online].

[19] Transparent Huge Pages in 2.6.38. http://lwn.net/Articles/423584/. [Online].

[20] H. Alam, T. Zhang, M. Erez, and Y. Etsion. Do-it-yourself virtual memory translation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 457–468, New York, NY, USA, 2017. ACM.

[21] T. E. Anderson, H. M. Levy, B. N. Bershad, and E. D. Lazowska. The interaction of architecture and operating system design. *SIGARCH Comput. Archit. News*, 19(2):108–120, Apr. 1991.

[22] D. N. Armstrong, Hyesoon Kim, O. Mutlu, and Y. N. Patt. Wrong path events: Exploiting unusual and illegal program behavior for early misprediction detection and recovery. In *37th International Symposium on Microarchitecture (MICRO-37'04)*, pages 119–128, Dec 2004.

[23] G. Ayers, J. H. Ahn, C. Kozyrakis, and P. Ranganathan. Memory hierarchy for web search. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 643–656, Feb 2018.

[24] J.-L. Baer and T.-F. Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Trans. Comput.*, 44(5):609–623, May 1995.

[25] 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)*, pages 399–411, Feb 2019.

[26] T. W. Barr, A. L. Cox, and S. Rixner. Translation Caching: Skip, Don'T Walk (the Page Table). In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 48–59, New York, NY, USA, 2010. ACM.

[27] T. W. Barr, A. L. Cox, and S. Rixner. SpecTLB: A Mechanism for Speculative Address Translation. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 307–318, New York, NY, USA, 2011. ACM.

[28] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift. Efficient virtual memory for big memory servers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 237–248, New York, NY, USA, 2013. ACM.

[29] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating two-dimensional page walks for virtualized systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 26–35, New York, NY, USA, 2008. ACM.

[30] A. Bhattacharjee. Large-reach memory management unit caches. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 383–394, New York, NY, USA, 2013. ACM.

[31] A. Bhattacharjee, D. Lustig, and M. Martonosi. Shared Last-level TLBs for Chip Multiprocessors. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 62–63, Washington, DC, USA, 2011. IEEE Computer Society.

[32] A. Bhattacharjee and M. Martonosi. Inter-core Cooperative TLB for Chip Multiprocessors. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 359–370, New York, NY, USA, 2010. ACM.

[33] J. Cavazos and D. Stefanovic. Adaptive prefetching using neural networks, 1997.

[34] G. Cox and A. Bhattacharjee. Efficient address translation for architectures with multiple page sizes. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 435–448, New York, NY, USA, 2017. ACM.

[35] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[36] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[37] A. Falcon, J. Stark, A. Ramirez, K. Lai, and M. Valero. Prophet/critic hybrid branch prediction. In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, pages 250–261, June 2004.

[38] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 37–48, New York, NY, USA, 2012. ACM.

[39] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 209–217, New York, NY, USA, 1998. ACM.

[40] N. Ganapathy and C. Schimmel. General purpose operating system support for multiple page sizes. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '98, pages 8–8, Berkeley, CA, USA, 1998. USENIX Association.

[41] A. Glew. Branch and computation refinement. 09 2019.

[42] S. Haria, M. D. Hill, and M. M. Swift. Devirtualizing memory in heterogeneous systems. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 637–650, New York, NY, USA, 2018. ACM.

[43] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. E. Kozyrakis, and P. Ranganathan. Learning memory access patterns. *ArXiv*, abs/1803.02329, 2018.

[44] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sept. 2006.

[45] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *2008 International Symposium on Computer Architecture*, pages 39–50, June 2008.

[46] E. Jacobsen, E. Rotenberg, and J. E. Smith. Assigning confidence to conditional branch predictions. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 29, pages 142–152, Washington, DC, USA, 1996. IEEE Computer Society.

[47] A. Jain and C. Lin. Rethinking belady's algorithm to accommodate prefetching. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 110–123, June 2018.

[48] D. A. Jiménez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 243–, Washington, DC, USA, 2003. IEEE Computer Society.

[49] D. A. Jiménez. Reconsidering complex branch predictors. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pages 43–52, Feb 2003.

[50] D. A. Jiménez. Piecewise linear branch prediction. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA '05, pages 382–393, Washington, DC, USA, 2005. IEEE Computer Society.

[51] D. A. Jiménez. An optimized scaled neural branch predictor. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 113–118, Oct 2011.

[52] D. A. Jiménez. Insertion and promotion for tree-based pseudolru last-level caches. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 284–296, Dec 2013.

[53] D. A. Jiménez, S. W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pages 67–76, Dec 2000.

[54] D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 197–, Washington, DC, USA, 2001. IEEE Computer Society.

[55] D. A. Jiménez and C. Lin. Perceptron learning for predicting the behavior of conditional branches. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 3, pages 2122–2127 vol.3, July 2001.

[56] D. A. Jiménez and G. H. Loh. Controlling the power and area of neural branch predictors for practical implementation in high-performance processors. In *2006 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'06)*, pages 55–62, Oct 2006.

[57] D. A. Jiménez and E. Teran. Multiperspective reuse prediction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 436–448, New York, NY, USA, 2017. ACM.

[58] R. Kalla, B. Sinharoy, and J. M. Tendler. Ibm power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, Mar. 2004.

[59] G. B. Kandiraju and A. Sivasubramaniam. Going the Distance for TLB Prefetching: An Application-driven Study. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, pages 195–206, Washington, DC, USA, 2002. IEEE Computer Society.

[60] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks. Profiling a warehouse-scale computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 158–169, New York, NY, USA, 2015. ACM.

[61] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal. Redundant memory mappings for fast access to large memories. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 66–78, New York, NY, USA, 2015. ACM.

[62] V. Karakostas, J. Gandhi, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. S. Unsal. Energy-efficient address translation. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 631–643, March 2016.

[63] V. Karakostas, O. S. Unsal, M. Nemirovsky, A. Cristal, and M. M. Swift. Performance analysis of the memory management unit under scale-out workloads. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–12, 2014.

[64] S. Khan and D. A. Jiménez. Insertion policy selection using decision tree analysis. In *2010 IEEE International Conference on Computer Design*, pages 106–111, Oct 2010.

[65] S. M. Khan, Y. Tian, and D. A. Jiménez. Sampling dead block prediction for last-level caches. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 175–186, Dec 2010.

[66] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, March 2005.

[67] R. Kumar, B. Grot, and V. Nagarajan. Blasting through the front-end bottleneck with shotgun. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 30–42, New York, NY, USA, 2018. ACM.

[68] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–480, Dec 2009.

[69] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, pages 694–701, Piscataway, NJ, USA, 2011. IEEE Press.

[70] S.-w. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou. Machine learning-based prefetch optimization for data center applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 56:1–56:10, New York, NY, USA, 2009. ACM.

[71] L. Liu. Multiple-page translation for tlb. In *Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93*, pages 344–349, Oct 1993.

[72] N. R. Mahapatra and B. V. Venkatrao. The processor-memory bottleneck: problems and solutions. In *CROS*, 1999.

[73] M. Minsky and S. Papert. Perceptrons - an introduction to computational geometry. 1969.

[74] S. Mirbagher Ajorpaz, E. Garza, S. Jindal, and D. A. Jiménez. Exploring predictive replacement policies for instruction cache and branch target buffer. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 519–532, June 2018.

[75] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[76] J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, transparent operating system support for superpages. In *Proceedings of the 5th Symposium on Operating Systems Design and implementation-Copyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading*, OSDI '02, pages 89–104, Berkeley, CA, USA, 2002. USENIX Association.

[77] M. Papadopoulou, X. Tong, A. Seznec, and A. Moshovos. Prediction-based superpage-friendly TLB designs. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 210–222, Feb 2015.

[78] C. H. Park, T. Heo, J. Jeong, and J. Huh. Hybrid TLB Coalescing: Improving TLB Translation Coverage Under Diverse Fragmented Memory Allocations. ISCA 2017.

[79] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[80] L. Peled, S. Mannor, U. Weiser, and Y. Etsion. Semantic locality and context-based prefetching using reinforcement learning. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 285–297, June 2015.

[81] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder. Using simpoint for accurate and efficient simulation. *SIGMETRICS Perform. Eval. Rev.*, 31(1):318–319, June 2003.

[82] B. Pham, A. Bhattacharjee, Y. Eckert, and G. H. Loh. Increasing TLB reach by exploiting clustering in page translations. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 558–567, Feb 2014.

[83] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee. CoLT: Coalesced Large-Reach TLBs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 258–269, Washington, DC, USA, 2012. IEEE Computer Society.

[84] B. Pham, J. Veselý, G. H. Loh, and A. Bhattacharjee. Large pages and lightweight memory management in virtualized environments: Can you have it both ways? In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 1–12, New York, NY, USA, 2015. ACM.

[85] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod. Using the simos machine simulator to study complex computer systems. *ACM Trans. Model. Comput. Simul.*, 7(1):78–103, Jan. 1997.

[86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[87] J. H. Ryoo, N. Gulur, S. Song, and L. K. John. Rethinking TLB Designs in Virtualized Environments: A Very Large Part-of-Memory TLB. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 469–480, New York, NY, USA, 2017. ACM.

[88] A. Saulsbury, F. Dahlgren, and P. Stenström. Recency-based TLB Preloading. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, pages 117–127, New York, NY, USA, 2000. ACM.

[89] A. Seznec. Concurrent support of multiple page sizes on a skewed associative TLB. *IEEE Transactions on Computers*, 53(7):924–927, July 2004.

[90] R. St. Amant, D. A. Jiménez, and D. Burger. Low-power, high-performance analog neural branch prediction. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 447–458, Nov 2008.

[91] M. Swanson, L. Stoller, and J. Carter. Increasing tlb reach using superpages backed by shadow memory. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, pages 204–213, Washington, DC, USA, 1998. IEEE Computer Society.

[92] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson. Tradeoffs in supporting two page sizes. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA '92, pages 415–424, New York, NY, USA, 1992. ACM.

[93] E. Teran, Y. Tian, Z. Wang, and D. A. Jiménez. Minimal disturbance placement and promotion. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 201–211, March 2016.

[94] E. Teran, Z. Wang, and D. A. Jiménez. Perceptron learning for reuse prediction. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49, pages 2:1–2:12, Piscataway, NJ, USA, 2016. IEEE Press.

[95] S. P. Vanderwiel and D. J. Lilja. Data prefetch mechanisms. *ACM Comput. Surv.*, 32(2):174–199, June 2000.

[96] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, Mar. 1995.

[97] I. Yaniv and D. Tsafrir. Hash, don't cache (the page table). In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, pages 337–350, New York, NY, USA, 2016. ACM.

[98] Y. Zeng and X. Guo. Long short term memory based hardware prefetcher: A case study. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '17, pages 305–311, New York, NY, USA, 2017. ACM.