Modern processors rely on predictive structures and policies (e.g., branch predictor, branch target buffer, cache prefetch and replacement policies, etc.) when making important control and data flow execution decisions [1, 2]. To maintain forward progress while awaiting resolving instructions, early computer architects looked toward speculation, acting on predicted directions of control flow. Without timely and accurate prediction, processor architectures might still be relegated to the age of stall-heavy sequential execution: now-basic computer architectural concepts like instruction-level parallelism, out-of-order execution, and speculative execution rely on making sound predictions. While we as humans may still not be able to look into the future, we have methods for "learning from the past" to make useful predictions of what is to come. Within the realm of computing, my research has focused on making microarchitectural predictive structures and their policies more accurate and resilient to changing computing demands and trends. The following sections overview some of my work on improving predictive structures and policies.

Indirect Branch Target Prediction

A major predictive structure along the critical path of execution is the branch target buffer, or BTB [3]. The BTB supports branch prediction and speculation by caching branching instructions' previously seen target address values.

The processor must also consider *indirect* branches, whose target addresses are unknown (i.e., not hardcoded in the instruction) and are computed at execution time. The BTB is insufficent to do this on-the-fly address computation; using it naively would result in unwanted control flow transfers and pipeline flushes from resulting misspeculations. Difficult-to-predict indirect branches have become increasingly common as object-oriented languages become more prevalent [4] in computing workloads[1].

This change in instruction trends motivated us to create a branch target prediction scheme that better predicts indirect branches. Using a series of small learning predictors, my work **Bit-Level Perceptron-Based Branch Predictor** [8], or BLBP, predicts individual branch target addresses' based on correlations in branch history. BLBP predicts target address values at the *bit level*, outputting individual `0` and `1` bit predictions for the low-order bits of indirect branch target addresses. BLBP then selects the BTB address entry which matches closest to the output bit-order predictions. BLBP achieves a misprediction rate of 0.183 misses per 1000 instructions (MPKI), compared to 0.193 MPKI for ITTAGE [9] and 0.29 MPKI for a VPC-based [4] indirect predictor.

BLBP helps the instruction front end remain fed with correctly speculated instruction streams. Though computing workloads' instruction mixes may evolve over time, schemes such as BLBP can help architectures remain agnostic to these trends, maintaining high processor performance.

Predictive Replacement Policies for Translation Lookaside Buffers

Translation Lookaside Buffers (TLBs) [10, 11] play a critical role in hardware-supported memory virtualization. To speed up address translation and reduce costly page table walks, TLBs cache a small number of recently used virtual-to-physical address translations. The TLB lies on the critical path to accessing memory and must make the best use of its limited capacities [12]. Thus, TLB entries with low potential for reuse should be replaced by more useful entries. We do this by evicting entries deemed

[1]Object-oriented languages generate many more indirect branches due to their support of polymorphism [5, 6]. For example, up to 23 times more indirect branches have been observed in C++ programs compared to those in C. [7]. Indirect branches are also ubiquitous since they support programming constructs like switch-case statements, jump tables, function pointer calls, and procedure returns.

dead, or unlikely to be re-referenced [13]. Replacement policies are also predictors – they predict the future usefulness of a cached entry and evict it if deemed dead.

Control Flow History Reuse Prediction [14], or CHiRP, contributes to an aspect of TLB management that has received little attention in the literature: replacement policy. To the best of our knowledge, the literature has only advocated least-recently-used (LRU) replacement or random replacement policies for TLBs and have proposed no TLB-specific replacement policy.

CHiRP shows how predictive replacement policies can be tailored toward TLBs, reducing miss rates and improving overall performance. By first applying recently proposed predictive cache replacement policies to the TLB, we show these policies do not currently work well without considering specific TLB behavior. This behavior stems mainly from the coarser granularity of TLB entries compared that of cache blocks, roughly 4KB versus 64B! CHiRP uses a history signature and replacement algorithm that correlates to TLBs' larger data granularities, outperforming other policies applied to TLBs. CHiRP reduces misses per 1000 instructions (MPKI) by an average 28.21% over the common LRU policy.

CHiRP is especially useful in the era of mixed page sizes, which can range from 4KB up to 1GB [15, 16]. Imagine: if one TLB entry's page covers 4KB of program data and another covers 2MB, which one's translation entry is more important to keep? The answer is not as easy as always choosing to keep larger pages for their data footprint/locality—the cost of keeping large pages in memory must also be considered, along with their known negative effects like memory fragmentation. On the other hand, smaller pages' translations are "cheaper" to maintain (and many more fit in memory!), but may not adequately cover larger applications memory footprints and lead to frequent page faults. The general caching problem [17] applied to TLBs suddenly becomes much more complicated, all because of supporting non-uniform sizes!

We have set the stage for further research by the computer architecture community into TLB replacement. As computing workloads trend ever larger [18, 19, 20], memory caching structures like the TLB are the first line of defense for avoiding costly memory accesses latencies.

Predictive Replacement Policies for the Instruction Cache and Branch Target Buffer

Just like translation lookaside buffers (TLBs) discussed in the previous section, both instruction caches (I-cache) and branch target buffers (BTBs) must make the best use of their limited capacities amidst the same timing constraints. Also like TLBs, replacement policies for the I-cache and TLB are a relatively unexplored topic, with little literature on the topic. Thus, we developed **Global History Reuse Prediction**, or GHRP [21], the first predictive replacement policy tailored for the I-cache and BTB. GHRP uses the history of past instruction and branch addresses to predict dead blocks eligible for eviction in the instruction cache and branch target buffer. We show that a highly accurate predictive replacement policy like GHRP can help reduce misses in the I-cache and BTB. GHRP reduces the rate of misses per 1000 instructions (MPKI) an average of 18% over a least-recently-used policy in over 600 tested workloads.

While coarse page granularity molded the history signatures used by CHiRP, GHRP looks to global histories of instruction and branch addresses for its signature generation. Previous known replacement policies are ill-suited for the I-cache and BTB because their tracking signatures are PC-based [22, 23], a feature found to have little correlation with instruction and branch reuse.

This work was a collaborative effort, influencing my interest and focus on predictive structures such as the BTB, which led to the generation of my BLBP work described before. As a motivating work and as a co-author, I present this as part of my cohesive story of predictive structure and policy improvement strategies.

Through the previous sections, I emphasize the need for smarter predictive structures and policies to

combat changing computing trends–the I-cache and BTB are obviously no exception to this. Along with the TLB, the I-cache and BTB help comprise the highly speculative front end of processors; keeping them highly accurate ensure better instruction throughput, no matter the workloads.

Current Work and The Future

My current as-yet-unpublished work continues the ideas presented in this statement. In particular, I now look toward creating more dynamic instruction and data prefetching structures via hybridization techniques. The act of prefetching is yet another form of computer architecture prediction. This time, it is the prediction of not just what instructions or data my program will soon require, but also when. We aim to increase prefetching coverage (i.e. data prefetched on time and found useful) by taking advantage of multiple prefetchers' learning techniques. Hybridization is known to be successful in other predictive structures, including prefetchers [24]. I would love to say more, but this work is not yet ready for publication!

While many teaching-focused professors tend to conduct research in computer science education, I wish to continue my work in computer architecture. Instruction and data working set sizes are significantly increasing, presenting a challenge to the way industry currently approaches microarchitecture design. My research vision for the next ten years is to investigate how best to leverage microarchitectural prediction to overcome this challenge, given real-world technology constraints. Collaborating with academic and industrial colleagues, I wish to advise undergraduate and graduate students drawn from a diverse population, offering novel solutions to tomorrow's technological problems while nurturing and training future generations of computer architects.

I believe it's important to remain current in my research: along with entry level courses, I especially hope to teach architecture-related courses. It is thus doubly important that my teaching reflects new trends and knowledge.

References

[1] A. Ramirez, O. J. Santana, J. L. Larriba-Pey, and M. Valero, "Fetching Instruction Streams," in *Proceedings of the 35th International Symposium on Microarchitecture*, (Istanbul, Turkey), December 2002.

[2] T.-Y. Yeh and Y. N. Patt, "A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution," in *ACM SIGMICRO Newsletter*, vol. 23, pp. 129–139, IEEE Computer Society Press, 1992.

[3] J. K. F. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, vol. 17, pp. 6–22, Jan 1984.

[4] H. Kim, J. A. Joao, O. Mutlu, C. J. Lee, Y. N. Patt, and R. Cohn, "VPC Prediction: Reducing the Cost of Indirect Branches via Hardware-based Dynamic Devirtualization," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, (New York, NY, USA), pp. 424–435, ACM, 2007.

[5] L. Cardelli and P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism," *ACM Comput. Surv.*, vol. 17, pp. 471–523, December 1985.

[6] H. Srinivasan and P. F. Sweeney, "Evaluating Virtual Dispatch Mechanisms for C++," Tech. Rep. Technical Report RC 20330, IBM TJ Watson Center, January 1996.

[7] B. Calder and D. Grunwald, "Reducing Indirect Function Call Overhead in C++ Programs," in *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, (New York, NY, USA), pp. 397–408, ACM, 1994.

[8] E. Garza, S. Mirbagher-Ajorpaz, T. A. Khan, and D. A. Jiménez, "Bit-Level Perceptron Prediction for Indirect Branches," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–38, 2019.

[9] A. Seznec, "A 64-Kbytes ITTAGE Indirect Branch Predictor," in *Proceedings of the JWAC-2: Championship Branch Prediction*, June 2011.

[10] J. F. Couleur and E. L. Glaser, "Shared-Access Data Processing System," November 19 1968. US Patent 3,412,382.

[11] D. W. Clark and J. S. Emer, "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement," *ACM Trans. Comput. Syst.*, vol. 3, pp. 31–62, February 1985.

[12] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[13] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-Block Prediction & Dead-Block Correlating Prefetchers," in *ACM SIGARCH Computer Architecture News*, vol. 29, pp. 144–154, ACM, 2001.

[14] S. Mirbagher-Ajorpaz, E. Garza, G. Pokam, and D. A. Jiménez, "CHiRP: Control-Flow History Reuse Prediction," in *To be presented at the 53rd International Symposium on Microarchitecture*, 2020.

[15] S. Srinivas, U. Pawar, D. Aribuki, C. Manciu, and G. Schulhof, "Runtime performance optimization blueprint: Intel architecture optimization with large code pages," Tech. Rep. Intel White Paper, https://software.intel.com/sites/default/files/managed/a0/0e/RuntimePerformanceOptimizationBlueprint_LargeCodePages.pdf, Intel, 2019.

[16] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson, "Tradeoffs in Supporting Two Page Sizes," in *Proceedings the 19th Annual International Symposium on Computer Architecture*, pp. 415–424, May 1992.

[17] S. Albers, S. Arora, and S. Khanna, "Page Replacement for General Caching Problems," in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, (USA), p. 31–40, Society for Industrial and Applied Mathematics, 1999.

[18] S. Kanev, J. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a Warehouse-Scale Computer," in *ISCA '15 Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 158–169, 2014.

[19] 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, (New York, NY, USA), pp. 37–48, ACM, 2012.

[20] 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)*, pp. 643–656, Feb 2018.

[21] S. Mirbagher-Ajorpaz, E. Garza, S. Jindal, and D. A. Jiménez, "Exploring Predictive Replacement Policies for Instruction Cache and Branch Target Buffer," in *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1-6, 2018*, pp. 519–532, 2018.

[22] S. M. Khan, Y. Tian, and D. A. Jiménez, "Sampling Dead Block Prediction for Last-Level Caches," in *MICRO*, pp. 175–186, December 2010.

[23] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, J. Simon C. Steely, and J. Emer, "SHiP: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, (New York, NY, USA), pp. 430–441, ACM, 2011.

[24] S. Kondguli and M. Huang, "Division of Labor: A More Effective Approach to Prefetching," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 83–95, 2018.