Statement of Research Interests

Yuxuan Zhang

As we have entered the Post-Moore Era where transistor size is reaching its limit, single-core performance can no longer be improved much. Although peripherals' performance and capacity have been dramatically increased and the concept of heterogeneous designs which allow the processor to offload part of its work to accelerators for speeding up program's execution have been proposed, the general purpose processor itself can still be a bottleneck for application's performance.

To address the inefficiency caused by processors, especially cache bottlenecks that remain despite decades of hardware innovations, I build systems that leverage both hardware and software techniques to improve application's performance. In these systems, hardware *monitors* performance and provides feedback to software, then software *optimizes* based on the feedback reported from hardware while applications are running. My past research includes two projects: (1) Online COde Layout Optimizations (Ocolos) [6] [7] and (2) Robust Profile-Guided Runtime Prefetch Generation (RPG^2) [10]. The former has shown it is feasible to apply profile-guided code layout optimization at runtime with low overhead. The latter shows that runtime optimization can also be applied to data cache prefetching if the code layout is carefully tuned to support on-stack replacement and continuous optimization. In this statement, I summarize my prior works, and detail my future research plans.

1 Prior Research - Profile-Guided Code Layout Optimization at Runtime

Online COde Layout Optimizations (Ocolos): Prior research [1] has pointed out that the capacity of the instruction-cache of server processors cannot handle the growth of today's data center applications and their growing instruction working set. Several profile-guided compiler-based approaches [2] [3] [5] have been proposed to address the gap between the instruction cache capacity and the code size. However, they all require halting the program to re-launch the optimized application. This is unacceptable for many services which handle thousands of queries or tasks per second, and these queries or tasks may change rapidly depending on how the inputs change. A newly optimized code layout may become stale after a relatively short period of time. Then it becomes necessary to change the code of the running process again once the system detects that the current code layout is not optimal for the incoming queries or tasks anymore.

To address this problem, I implemented Ocolos [6] [7], the first *online* code layout optimization system for unmodified applications written in unmanaged languages like C and C++. Ocolos allows profile-guided optimization to be performed on a running process, instead of being performed offline and requiring the application to be re-launched. By running online, profile data is always relevant to the current execution and always maps perfectly to the running code. Ocolos demonstrates how to achieve robust online code replacement in complex multithreaded applications like MySQL and MongoDB, without requiring any application changes. Our experiments show that OCOLOS can accelerate big-code applications such as MySQL by up to 1.41×.

Robust Profile-Guided Runtime Prefetch Generation (RPG^2): Data cache prefetching is a well-established problem. Since current major server processors employ only simple hardware prefetchers, which fail to capture complex indirect access patterns, a number of static data prefetch compilers [8] [9] have been proposed. However, as the data that needs to be prefetched depends on not only program structure, but also program input and the microarchitecture, it is hard for static compiler-based prefetch insertion approaches to provide a solution that can benefit to an application across all inputs. Moreover, depending on the input and the microarchitecture, prefetch instructions can sometimes be harmful to performance. It is hard for developers or compilers to know up-front if prefetching will help or hurt.

To make up for the shortcomings caused by static compiler-based prefetch insertion approaches, I designed and implemented the RPG^2 system for *online* prefetch injection and tuning. RPG^2 profiles a running C/C++ program, injects prefetch instructions and then tunes those prefetches to maximize performance. RPG^2 can provide a comparable speedup to the best profile-guided prefetching insertion compilers, but can also respond when prefetching ends up being harmful and roll back to the original code – something that static compilers cannot. In other words, RPG^2 makes prefetching a safer optimization by preserving its upsides, and protecting programs from its downsides. According to our experimental results, RPG^2 can provide speedup up to 2.15× in the case where prefetch can benefit to the performance. In static compiler-based prefetch insertion approach's worst case scenario where there is a 70% slowdown, the slowdown caused by RPG^2 is only 7%.

2 Future Research Agenda

PGO-based Code Prefetching at Runtime: My previous work [6] has shown that the performance of big-code applications is input-dependent. Hence, we need a system like Ocolos to dynamically tune the code layout for different inputs in order to achieve the best performance at runtime. Ocolos' runtime basic block reordering and function reordering significantly improve processor's front-end performance, because if frequently accessed basic blocks and functions are reordered to be adjacent, capacity and conflict misses of instruction cache caused by big-code applications will be reduced. However, reordering basic blocks and functions fall short for reducing compulsory misses. To enable more *online* optimizations, it's also necessary for Ocolos to support runtime code prefetching, since it helps with reducing the compulsory misses.

Prior work of profile-guided code prefetch insertion approaches [2] [3] [4] tried to address the performance degradation caused by instruction cache's compulsory misses *offline*, but at the time I implemented Ocolos, none of them can be adopted by Ocolos because of a lack of ISA support for code prefetch instructions. For example, to increase both code prefetch coverage and accuracy, I-SPY [3] proposed the concept of conditional prefetch. On their hardware simulator, a new code prefetch instruction `Cprefetch` was introduced to the x86 instruction set. The `Cprefetch` instruction integrates the trace of predecessor basic blocks that has a high probability of causing an instruction cache miss in a particular successor basic block. When `Cprefetch` is executed at runtime, the processor examines whether the last 32 basic blocks collected from intel's Last Branch Records (LBR) contain the trace encoded in `Cprefetch`, and then decides whether code should be prefetched. Of course, this `Cprefetch` instruction doesn't exist in any actual processors.

The good news is that new instructions being rolled out soon by Intel with its Redwood Cove microarchitecture will make software instruction prefetching possible. But with only the basic hardware code prefetching instructions added to the x86 ISA, it is not enough to reproduce the functionality proposed by any of the state of the art code prefetch solutions on real machines. So one of my future research goals is to seek an alternative way that leverages Intel's code prefetch instructions to implement I-SPY or other code prefetching strategies on real processors, and then convert it to be *online*. For instance, I-SPY's profiling, prefetch injection analysis and `Cprefetch`'s examination of whether the most recent 32 LBR samples contain the trace that causes instruction cache miss can all be moved to the new system. With the information of prefetch injection sites and traces that causes instruction cache misses from the profile, once the new system detects any/none of the traces exists in the current LBR samples, it can dynamically insert/erase code prefetch instructions. Since runtime code replacement has additional overhead, it would be interesting to see how much performance degradation will arise if the decisions of whether insert prefetch or not are made by the new system before executing each possible prefetch spot at runtime. It would also be interesting to know what the maximum performance improvement the new version of the new system can achieve if all runtime optimizations (i.e., basic block reordering, function reordering *and* code prefetching) are turned on.

References

[1] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G. Wei, D. Brooks, "Profiling a warehouse-scale computer," *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, Portland, OR, USA, 2015, pp. 158-169.

[2] G. A. Nayana, P. Nagendra, D. I. August, H. Cho, S. Kanev, C. Kozyrakis, T. Krishnamurthy, H. Litz, T. Moseley, P. Ranganathan, "AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale

Computers," *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, Phoenix, AZ, USA, 2019, pp. 462-473.

[3] T. A. Khan, A. Sriraman, J. Devietti, G. Pokam, H. Litz and B. Kasikci, "I-SPY: Context-Driven Conditional Instruction Prefetching with Coalescing," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Athens, Greece, 2020, pp. 146-159.

[4] T. A. Khan , D. Zhang, A. Sriraman, J. Devietti, G. Pokam, H. Litz, B.Kasikci, "Ripple: Profile-Guided Instruction Cache Replacement for Data Center Applications," *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, 2021, pp. 734-747.

[5] M. Panchenko, R. Auler, B. Nell and G. Ottoni, "BOLT: A Practical Binary Optimizer for Data Centers and Beyond," *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Washington, DC, USA, 2019 pp. 2-14.

[6] Y. Zhang, T. A. Khan, G. Pokam, B. Kasikci, H. Litz and J. Devietti, "OCOLOS: Online COde Layout OptimizationS," *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 530-545.

[7] Y. Zhang, T. A. Khan, G. Pokam, B. Kasikci, H. Litz and J. Devietti, "Online Code Layout Optimizations via OCOLOS," in *IEEE Micro, vol. 43, no. 4, pp. 71-79, "Top Picks From the 2022 Computer Architecture Conferences"*, July-Aug. 2023.

[8] S. Jamilan, T. A. Khan, G. Ayers, B. Kasikci, and H. Litz. 2022, "APT-GET: profile-guided timely software prefetching," *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22)*, New York, NY, USA, 747–764.

[9] T. A. Khan, I. Neal, G. Pokam, B. Mozafari, and B. Kasikci. "Dmon: Efficient detection and correction of data locality problems using selective profiling," *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), pages 163–181*. USENIX Association, July 2021.

[10] Y. Zhang, N. Sobotka, S. Park, S. Jamilan, T. A. Khan, B. Kasikci, G. Pokam, H. Litz, J. Devietti. "RPG2: Robust Profile-Guided Runtime Prefetch Generation" *In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), 999–1013.*