RETROSPECTIVE: Self-optimizing Memory Controllers: A Reinforcement Learning Approach

José F. Martínez
Cornell University
Ithaca, NY USA
martinez@cornell.edu

Engin Ipek
Qualcomm
San Diego, CA
eipek@qti.qualcomm.com

Onur Mutlu
ETH Zurich
Zurich, Switzerland
onur.mutlu@inf.ethz.ch

Rich Caruana
Microsoft Research
Redmond, WA USA
rcaruana@microsoft.com

I. The path to reinforcement learning

Around the time we began working on this idea, the computer architecture community was not particularly familiar with machine learning (ML), and solutions to both offline (i.e., design- or compile-time) and online (i.e., run-time) problems were largely based on ad hoc heuristics. Some proposals did employ ML offline, including genetic algorithms for automatic derivation of branch predictor designs [O-14] or artificial neural networks (ANNs) for static branch prediction [O-7].[1] But when it came to online heuristics, ML-based solutions were virtually inexistent, with a notable exception in Jiménez and Lin's perceptron-based branch predictor design [O-19]. A perceptron is essentially a very simple ANN, and thus a form *supervised learning*, a.k.a. "learning with a teacher." Supervised learning matches the branch prediction problem very well, as the outcome of every branch can be used to train the perceptron on what its prediction should have been for that branch. Jiménez and Lin's work was elegant and showed that the perception could outperform the best-known ad hoc heuristics for that problem.

At Cornell University, Engin Ipek had taken Rich Caruana's course in ML and had become enamored with the technology and its possibilities. With collaborators Sally McKee, Bronis de Supinski, and Martin Schulz, Engin and Rich had successfully devised an offline methodology to dramatically cut down the exploration of a microprocessor's design space, using ANNs to pick what configurations to simulate [5]. José Martínez was impressed by this work; however, he was more excited about the potential of inserting an ML engine right into the hardware that could beat the best-known online ad hoc heuristics for key architectural mechanisms, as Jiménez and Lin had done for branch prediction.

Engin and José had been working together on the Core Fusion project [4] and wrestled with a problem that looked nothing like branch prediction: instruction steering. Instruction steering is a problem typical of clustered microarchitectures in general, and Core Fusion in particular: given a group of consecutively fetched instructions, which ones should be sent for execution to what core? Unlike branch prediction, steering decisions don't have an obvious "correct answer," even a posteriori, so supervised learning did not seem like a good fit. It is, in fact, a fairly complex problem because 1) each decision needs to balance multiple constraints (e.g., try to steer producers and consumers to the same core but, at the same time, try to balance the load across cores), and 2) the optimality of a decision depends in part on future unknowns (e.g., will an instruction have many data dependencies with others yet to be fetched, and if so, will we come to regret a decision based solely on current knowledge?). A number of key architecture problems share these two characteristics; instruction scheduling in superscalar processors is another example.

Engin, José, and Rich began to look for ML mechanisms that would be a good fit for instruction steering and scheduling, and soon enough they found promise in reinforcement learning (RL) [O-42]. An RL agent has an a priori understanding of how nominally "good" an action can be. This a priori knowledge is often hardwired into the RL agent (e.g., based on a computer architect's intuition). For example, in instruction scheduling, filling an issue slot is nominally "good," while leaving it unassigned is "bad"—at least in principle. However, when making a decision, RL looks at the system's current state and picks among a set of available actions not based on the perceived immediate reward, but on the estimated long-term effect of taking such an action. This was, fundamentally, an exciting twist: don't be preoccupied with what action seems "best" at each point in time; instead, keep your eyes on the ultimate prize—in our case, minimizing execution time. As the system evolves, based in part on the actions already taken, an RL agent has the capacity to progressively refine its ability to estimate the long-term effect of a particular action in a state similar enough to one visited before. This is called "learning through interaction."

RL based on Q values, in particular, was particularly attractive because it seemed readily implementable in silicon. A Q-value matrix is essentially a multidimensional table that is indexed periodically to 1) read the current best estimation of the long-term effect of taking a particular action in a particular system state, and 2) update the Q-value matrix using newly acquired hindsight as the system evolves. This sounded a lot like something that could be folded into good ol' RAM!

II. A solution in search of a problem

Designing an RL agent for instruction steering and scheduling turned out to be easier said than done, however. A particularly tough challenge was the fact that these operations were on the critical path of a microprocessor's datapath, and designing an RL mechanism that would access a Q-value matrix multiple times, once per possible action, before deciding which action to perform, was likely to take a handful of precious CPU cycles. José asked Engin: where else in a computer system could one find a scheduling problem that, on the one hand, had real impact on execution time, but on the other hand, had built-in slack to insert something as sophisticated as an RL agent? A few days later, Engin came up with something that seemed to have great potential: a memory controller.

A primary function of memory controllers is to schedule memory requests in a way that minimizes execution time. The

[1] Citations with an O- prefix refer to those in the original paper.

memory controller had long moved on-chip and was therefore clocked at CPU speed. DDR2 DRAM, on the other hand, was clocked at a fraction of that speed. This meant that, for each memory scheduling decision, an RL-based memory controller would have multiple CPU cycles to ponder options. The "golden standard" of memory scheduling was arguably FR-FCFS by Rixner et al. [O-37], and it was a perfect example of a great design that was heavily reliant on expert intuition and without the capacity to improve at run-time. The stage was set, then: The goal would be to beat FR-FCFS by taking a fundamentally different approach based on RL.

III. Designing the RL-based memory scheduler

Engin, José, and Rich (well, mostly Engin) began to dissect the problem into the standard RL components: actions, immediate rewards, and state attributes. Actions were dictated by what a memory controller could do (e.g., issue a read command or activate a row), so there wasn't much to decide there. For immediate rewards, our first try turned out to be quite good: +1 if an action was actually reading or writing data, 0 otherwise. (Janani Mukundan and José would conclude later that better immediate reward values could be obtained offline using genetic algorithms [8].) State attributes were tricky: it was important to sense the ones that best informed the RL agent, and it couldn't be too many, as the number had a direct bearing on the size and effectiveness of the Q-value matrix—a problem known in the ML community as the "curse of dimensionality," which in our case would materialize in the form of more silicon area and longer access times. Engin decided to take a systematic approach, using linear feature selection to narrow down a list of more than 200 attributes that could intuitively be connected to memory scheduling decisions. Importantly, he included attributes that were observable on chip but not necessarily in the memory controller itself (e.g., relative ROB order of memory requests). In fact, among the final six attributes, two of them would be of this kind, so the final design included a description of how one would bus this information along with a memory request.

With the basic design out of the way, we still had to find a way to fit the scheduler's decision-making process into the number of CPU cycles that constituted a single memory cycle for DDR2-800 DRAM and, of course, evaluate the design in a detailed simulator. Around that time, José visited Microsoft Research and met Onur Mutlu, who had a long record of publishing on architectures for memory subsystems and specifically on memory controller design. Shortly afterward, Engin joined Microsoft Research for a co-op under Onur, and it was during that time that the paper really came together. Already convinced that memory controllers need to become more intelligent and having implemented perceptron-based branch predictors himself, Onur bought into the idea immediately, and the whole team spent several months nailing down the design, implementation, detailed experiments, comparisons, and results analyses.

IV. The reviewers chime in

The paper was first submitted to MICRO in 2007. Reviewers rejected it, in the process posing several probing questions that would eventually make the final evaluation much stronger. Two really intriguing ones were: 1) Could the baseline FR-FCFS design have benefited from factoring in some of the additional attributes that feature selection found? 2) Could an offline RL-based design (i.e., one where the Q-value matrix is frozen after offline training through simulation and then installed in the final design) match your results? We carried out additional experiments to try these scenarios and were able to show that an online RL approach was still significantly superior to those two. The paper was highly reviewed and accepted to ISCA 2008.

V. The aftermath

Shortly after publishing the paper, RL pioneer Andrew Barto (UMass-Amherst) and co-author of the seminal book on RL [O-42] with Richard Sutton (U. of Alberta) reached out to us, as he had come across the work and thought was fascinating. The first edition of Sutton & Barto was a frequent source of inspiration and knowledge during our work, so we were incredibly flattered by this! Eventually, as Andrew and Richard were preparing the second edition of their book, Andrew reached out again to tell us that they had decided to include our work as one of eight case studies of successful uses of RL alongside projects like Google's AlphaGo and Watson's Jeopardy!

The design inspired follow-up work by us and many others: at the time of this writing, this paper has been cited over six hundred times. We believe this is in part because the problem was important and the ML-based approach inspiring, and in part because ML over the last ten years has become a major source of interest in computer architecture and beyond. As for the particular implementation itself, it is probably fair to say that it did not age well, as DRAM standards continued to crank up the clock while CPU frequency mostly stagnated, which essentially squeezed the clock margin that the original design enjoyed [3], [10]. However, we believe the fundamentals of the RL-based design can be applied and extended to the faster memory controllers of today and the future, and to other computer architecture problems as well [1], [2], [6], [7], [9], [11].

References

[1] Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *Intl. Symp. on Microarchitecture (MICRO)*, 2021

[2] Zhuo Chen and Diana Marculescu. Distributed reinforcement learning for power limited many-core system performance optimization. In *Design, Automation & Test in Europe Conf. (DATE)*, 2015

[3] Saugata Ghose, Hyodong Lee, and José F. Martínez. Improving memory scheduling via processor-side load criticality information. In *Intl. Symp. on Computer Architecture (ISCA)*, June 2013

[4] Engin İpek, Meyrem Kırman, Nevin Kırman, and José F. Martínez. Core Fusion: Accommodating software diversity in chip multiprocessors. In *Intl. Symp. on Computer Architecture (ISCA)*, 2007

[5] Engin İpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. Efficiently exploring architectural design spaces Via predictive modeling. In Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2006

[6] Wonkyung Kang and Sungjoo Yoo. Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in SSD. In *Design Automation Conference (DAC)*, 2018

[7] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. ConfuciuX: Autonomous hardware resource assign- ment for DNN accelerators using reinforcement learning. In *Intl. Symp. on Microarchitecture (MICRO)*, 2020

[8] Janani Mukundan and José F. Martínez. MORSE: Multi-objective reconfigurable self-optimizing memory scheduler. In *Intl. Symp. on High-Performance Computer Architecture (HPCA)*, 2012

[9] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. Semantic locality and context-based prefetching using reinforcement learning. In *Intl. Symp. on Computer Architecture (ISCA)*, 2015

[10] Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu. BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling. *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, Vol. 27, No. 10, 2016

[11] Hao Zheng and Ahmed Louri. An energy-efficient network-on-chip design using reinforcement learning. In *Design Automation Conference (DAC)*, 2019