FlexCPU: A Configurable Out-of-Order CPU Abstraction

Bradley Wang, Ayaz Akram, Jason Lowe-Power
University of California, Davis
{radwang, yazakram, jlowepower}@ucdavis.edu

***Abstract*—We present `FlexCPU`, a new software model for CPU performance integrated into gem5. `FlexCPU` combines the benefits of trace-based models with execute-in-execute semantics which leads to more accurate simulation of multithreaded and full-system applications. Our design is heavily inspired by dataflow models, and it reduces modern out-of-order techniques to abstracted parameterized constraints. `FlexCPU` can be configured to match the behaviors of modern general purpose CPUs and used for limit studies. By reducing CPU behaviors to reasonable abstractions and stages, `FlexCPU` is simpler to understand and easier to extend than other execute-in-execute CPU models. We show that `FlexCPU` can achieve the maximum theoretical ILP for most workloads and show a case study of using `FlexCPU` to model multiple processor architectures.**

I. Introduction

In this paper, we present a new CPU model that we integrate into gem5 [1] that we call `FlexCPU`. `FlexCPU` is an execute-in-execute model that takes inspiration from trace-based and dynamic binary translation model designs (e.g. Sniper [2], ZSim [3], and many others). It combines the flexibility trace-based models with the precision of execute-in-execute models. `FlexCPU` does this by separating the execution logic from the timing model so that the execution logic is unhampered by specific hardware design constraints. The execution logic implements out-of-order features, such as the instruction window, register renaming, speculative behavior, etc. Behaviors can then be *constrained* by the user with simple and easy to understand parameters which allows `FlexCPU` to model a wide range of hardware implementations from speculative out-of-order processors to in-order designs to future microarchitectures that can exploit high ILP.

When constructing CPU performance models there is a tradeoff between precisely modeling one specific microarchitecture and flexibility. Most current simulation systems pick one of the two extremes. For example, gem5's out-of-order CPU model (O3CPU) is a detailed model "with a strong focus on ensuring timing accuracy" originally designed for the Compaq Alpha XP1000 with an Alpha 21264 processor [4]. On the other hand, simulators like ZSim [3] and Sniper [2] give up execute-in-execute semantics and the increased execution-stream specificity that comes with models like O3CPU but gain significant flexibility.

Different simulation studies require different kinds of CPU models, and not all simulation studies should use a model "with a strong focus on ensuring timing accuracy" to a particular hardware implementation. Many studies want a CPU model that is forward-looking and can simulate *future* CPU designs. For instance, when studying novel memory systems, the details of any one CPU microarchitecture are often less important than its theoretical peak performance. Especially with the end of Dennard scaling and Moore's Law and the emergence of many on-chip accelerators, the computer architecture community needs a simple, flexible, easy to configure and understand out-of-order core model.

`FlexCPU` presents a new point on the precision-flexibility spectrum for architectural simulators. `FlexCPU` is almost as configurable as perfect knowledge trace-based simulation, while fully respecting execute-in-execute semantics. Thus, `FlexCPU` is applicable for performing complex simulations with I/O, full system kernel, and multithreaded workloads. It serves as a more general CPU than O3, appropriate for studies which are not about core microarchitecture specifics.

II. Use-cases

There are three main use-cases for `FlexCPU` that are not well served by current CPU models.

Conducting limit studies of ILP

By having a CPU model that is not constrained by a specific hardware implementation, it allows researchers to perform limit studies (at least within the bounds of execute-in-execute). `FlexCPU` can be used with full systems simulation, multiple cores, and with I/O, unlike existing trace-based and binary translation-based tools (e.g., MICA [5]).

Emulating future high performance out-of-order cores

When looking at future systems, we want to be able to simulate future out-of-order cores. However, simulating current or older microarchitecture limits CPU performance. With `FlexCPU`, researchers get to choose how aggressive they believe future CPUs will be and are not limited by today's microarchitecture implementation.

Modelling current processors accurately and simply

Finally, often when performing research on non-CPU microarchitecture components researchers simply want to have a "reasonable" CPU model. However, configuring gem5's rigid O3CPU to a "reasonable" system that models current processors can be challenging. For example, there is a paper dedicated to configuring gem5 to match an ARM SoC [6], and many papers showing gem5's shortcomings e.g. [7], [8].

FlexCPU is "relatively accurate," i.e. when you change a parameter (e.g., the fetch width) the processor's performance changes accordingly. Models with 100s of design specific parameters are not well suited for "relative accuracy" since it is not clear to the user how each parameter will affect performance. This makes FlexCPU useful as a teaching tool, as its constraints are simple and directly affect the performance in understandable ways.

III. Design Philosophy

In designing FlexCPU, we placed an emphasis on making a CPU whose fundamental behavior was straightforward. As a result, we present FlexCPU as a reduction of the overall behaviors of any CPU to a set of *dependencies* and *constraints*. As evidenced by FlexCPU's ability to model multiple classes of CPUs simply by adding or removing dependencies and constraints, we have shown that this reduction creates a powerful tool because we can make small, definite changes to cause understandable changes in behavior.

We believe that the simple CPU design of FlexCPU gives researchers a number of advantages over alternative designs:

1) **Easier to understand** if key ideas are just logical steps instead of specific implementations and complex coupling between hardware units.
2) **Easier to use** with simple logical parameters.
3) **Easier to extend** since we have a simple request-to-callback mechanism by which all structurally constrained steps are completed, and a simple stage-depends-on-stage mechanism by which logically constrained tasks are ordered.
4) **Easier to model** novel behaviors due to simple breakdown of stages between instructions, and a callback-driven dependency system to schedule events not at a specific time, but as early as certain conditions are met.

FlexCPU should become a good teaching/learning tool, on top of being a configurable research tool.

IV. Experiments and Results

As an example, we present a limit study case (Figure 1) where we observe that FlexCPU's performance is closer to the theoretical limits calculated by MICA [5] than O3 is capable of achieving. MICA, as a trace-based simulator, simulates under some assumptions which are not possible to make with execute-in-execute, such as having perfect branch prediction. We configure our systems to be as close to perfect as possible within execute-in-execute, including external MICA assumptions, such as ideal single-cycle memory. The microbenchmarks used for testing purposes are the ones that were used to validate another microarchitecture simulator SiNUCA [9].

For the microbenchmarks bound purely by true dependencies between instructions (e.g. chain1/chain6) we observe similar performance across the board. However, for microbenchmarks that stress the more inflexible structures in O3 such as the fetch, branch prediction, and LSQ units, FlexCPU is more capable of reaching theoretical limits by configuring

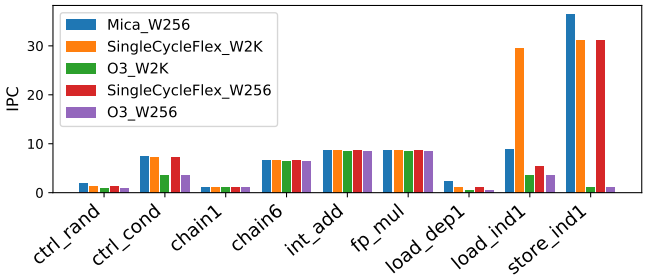

Fig. 1. Some IPC numbers between FlexCPU, O3, and MICA's theoretical limit analysis, for selected microbenchmarks. Note the different window sizes.

penalties to zero-time events. In the case of independent loads, since window-size is the dominating factor instead of true dependencies, FlexCPU at a window-size of 2048 reasonably manages to find more parallelism than MICA at a window size of 256.

V. Conclusion and Future Work

We believe that by starting with an ideal dataflow out-of-order model, we will be able to model any CPU by adding constraints. Future work will explore adding constraints to FlexCPU to model specific real-world hardware (e.g., Intel i7).

When compared to ideal trace-based systems, control dependencies such as branch mis-speculations are the main cause of reduced IPC in FlexCPU. We are currently investigating adding support for multiple branch paths, since the definition of a committed architectural state and streams of state transition readily lends itself to branches off the stream. The simplicity of selecting one and discarding the other should mean that exploration of multiple paths should be straightforward, and can get us even closer to an ideal scenario within the limits of execute-in-execute.

We plan to introduce FlexCPU into upstream gem5.

References

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, May 2011.

[2] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM TACO*, vol. 11, no. 3, Article 28, 2014.

[3] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, vol. 41, pp. 475–486, Tel-Aviv, Israel, Tel-Aviv, Israel, 23-27 June 2013.

[4] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, pp. 52–60, July/August 2006.

[5] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *IEEE Micro*, vol. 27, pp. 63–72, May 2007.

[6] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver, "Sources of Error in Full-System Simulation," in *ISPASS*, pp. 13–22, Monterey, CA, 2014.

[7] T. Nowatzki, J. Menon, C. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," vol. 35, pp. 4–12, 2015.

[8] A. Akram and L. Sawalha, "x86 computer architecture simulators: A comparative study," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pp. 638–645, IEEE, 2016.

[9] M. A. Z. Alves, C. Villavieja, M. Diener, F. B. Moreira, and P. O. A. Navaux, "Sinuca: A validated micro-architecture simulator," in *IEEE HPCC*, pp. 605–610, 2015.