Student Research Poster: Software Out-of-Order Execution for In-Order Architectures

Kim-Anh Tran
Department of Information Technology
Uppsala University
kim-anh.tran@it.uu.se

ABSTRACT

Processor cores are divided into two categories: fast and power-hungry out-of-order processors, and efficient, but slower in-order processors. To achieve high performance with low-energy budgets, this proposal aims to deliver out-of-order processing by software (SWOOP) on in-order architectures.

Problem: A primary cause for slowdown in in-order processors is last-level cache misses (caused by difficult to predict data-dependent loads), resulting in cores stalling.

Solution: As loads are non-blocking operations, independent instructions are scheduled to run before the loads return. We execute critical load instructions earlier in the program for a three-fold benefit: increasing memory and instruction level parallelism, and hiding memory latency.

Related work: Some instruction scheduling policies attempt to hide memory latency, but scheduling is confined by basic block limits and register pressure. Software pipelining [3] is restricted by dependencies between instructions and decoupled access-execute (DAE) [1] suffers from address re-computation. Unlike EPIC [2] (evolved from VLIW), SWOOP does not require hardware support for predicated execution, speculative loads and their verification, delayed exception handling, memory disambiguation etc.

Keywords

Software Out-of-Order Execution; Energy; Compiler

1. THE SWOOP COMPILER

Contribution: (1) *Overcoming dependent loads.* SWOOP detects dependent load chains and splits them up into multiple access phases, enabling independent instructions to be scheduled in-between. (2) *Handling of unknown memory dependencies.* May-aliasing loads are turned into safe prefetches. (3) *Balancing of latency and register pressure.* SWOOP selectively turns safe loads to prefetches to control register pressure. We transform the code using LLVM in a way similar to DAE. Figure 1 shows the transformation and execution. SWOOP (1) hoists critical loads, (2) creates multiple access phases to maximize separation, (3) turns may-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PACT '16 September 11-15, 2016, Haifa, Israel

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4121-9/16/09.

DOI: <http://dx.doi.org/10.1145/2967938.2971466>

Figure 1: SWOOP Transformation.

aliasing loads to prefetches to overcome unknown dependencies, and (4) inserts a *ckmiss* instruction. On a cache miss in an access phase, the *ckmiss* will trigger a branch to the next iteration's access phase. We introduce a register remapping technique to ensure that execute phases consume registers of their respective access phases. Figure 2 shows the

Figure 2: Normalized speed up.

normalized speedup of memory-bound benchmarks (SPEC 2006CPU, CIGAR, NAS) evaluated with the Sniper Simulator. We compare an in-order core (InO~Cortex-A7), the SWOOP core (InO with register remapping) and an out-of-order core (OoO~Cortex-A15), and achieve an average improvement of 34% in runtime and a reduction by 23% in energy consumption (not shown) over the InO.

2. ACKNOWLEDGMENTS

I would like to thank my advisors Alexandra Jimborean and Stefanos Kaxiras for their guidance and support.

3. REFERENCES

- [1] A. Jimborean et al. Fix the code. don't tweak the hardware: A new compiler approach to voltage-frequency scaling. In *CGO*, 2014.
- [2] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, Appendix H*. Morgan Kaufmann Publishers Inc., 2011.
- [3] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *PLDI*, 1988.