Value-Aware Scheduler for Energy Reduction

Haiyue Ma
Electrical & Computer Engineering
Princeton University
Princeton, NJ, USA
hm1@princeton.edu

Kaifeng Xu
Electrical & Computer Engineering
Princeton University
Princeton, NJ, USA
kaifengx@princeton.edu

David Wentzlaff
Electrical & Computer Engineering
Princeton University
Princeton, NJ, USA
wentzlaf@princeton.edu

***Abstract*—Optimizing energy consumption for computing workloads is a critical challenge, especially as we push the limits of high-performance systems within resource constraints. An underexplored approach to reducing energy costs is exploiting *value locality*: the tendency of instructions to process similar operand values during execution. Instruction operand values have abundant similarities within the reachable instruction window, a factor that existing architectures currently ignore. This paper introduces the broad concept of instruction-level, value-aware scheduling, an approach that leverages value locality in the hardware scheduler. By redirecting instructions of similar operand values to move together down the pipeline, the scheduler can reduce switching activity and dynamic energy usage. Such value-aware design philosophy, capable of adapting to a diverse range of workloads and architectures, has potential to be applied in broader architectural improvements.**

I. Introduction

Energy efficiency is a critical concern in modern computing [1]. While static energy usage remains relatively constant, dynamic energy usage is tied to the circuit switching activity (the number of bit flips) during program execution and presents opportunities for optimization. By executing programs in a manner that reduces switching activity, we can save dynamic energy without affecting the result and the performance of the program.

One promising program feature that can be leveraged for energy reduction is value locality. Value locality refers to similar instruction operand values occurring throughout the program. Since register transitions between similar values can lead to lesser switching activity compared to transitions between uncorrelated values, we can reduce dynamic energy consumption if we are operating on similar, **consecutive** values on the same data path.

Prior research has shown a high degree of operand value similarities observed at the hardware architecture level [2] [3]. However, most existing exploitations on value locality require either compiler support [4] [5] or fine-tuning in the program [3] [6]. Few have directly exploited this knowledge for dynamic optimization in hardware. This paper introduces the concept of **value-aware hardware scheduling** that can reduce energy consumption by scheduling instructions with similar operand values together, as illustrated in Figure 1. While state-of-the-art schedulers assign an instruction onto the first execution lane available, a value-aware scheduler strategically assigns instructions with similar operands to flow down the pipeline consecutively. This scheduling policy minimizes switching activities between instructions and can reduce dynamic energy without changing the program.

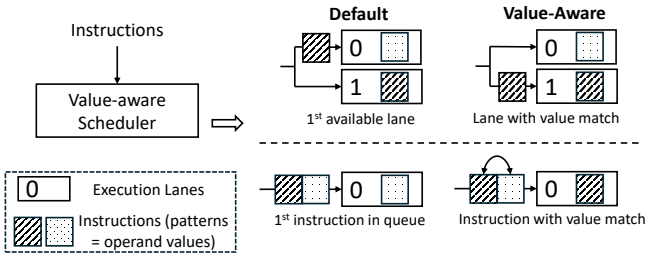


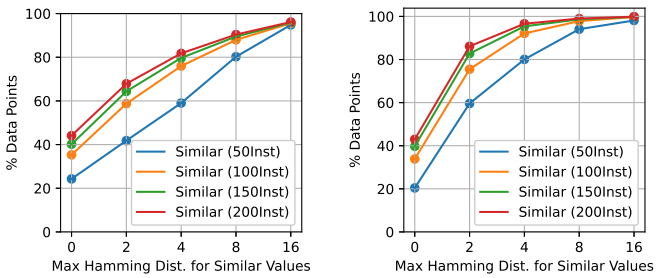
Fig. 1: Overall workflow for the proposed scheduling technique: a dynamic hardware scheduler that predicts instruction operand value similarities, and executes instructions with similar values consecutively on the same execution lane to reduce toggling.

II. Opportunities in Value Locality

Value locality can be classified into two types: **static value locality**, where one or a few values occur most of the time [2]; and **dynamic value locality**, where a value keeps changing but remains the same as the one generated elsewhere in the program [3]. Static value locality consists of values from constant sources or program regularities that are known at compile time [7] [8]. Dynamic value locality refers to variants coming from program inputs that vary from run to run and can only be known at runtime [3] [9] [10]. Although the program input is typically different across program runs, there is usually a consistent pattern of similarity between data inputs [6]. For example, edge applications [11] [12] and big data applications [13] [14] process data from real-world which are naturally correlated. The proposed scheduling technique has the potential to optimize both dynamic and static locality to get the benefits of both worlds.

Figure 2 shows value locality in common applications, including SPEC2017 [15][1], a general-purpose benchmark suite, and EdgeBench [16], a data-intensive edge application benchmark suite (averaged across all individual benchmarks). For each testbench, we ran a trace of 100k instructions after the warmup stage (skipping first one billion instructions) and counted the number of non-memory instructions that have the

[1]These SPEC 2017 results are non-compliant estimates.



(a) SPEC2017 (General Purpose). (b) EdgeBench (Data Intensive).

Fig. 2: Percentage of non-memory instructions that have the first source operand value match with at least one other instruction within various instruction window lengths. Both benchmark suites show significant value locality; EdgeBench shows more data input similarities while SPEC2017 shows more program-related value correlation.

first source operand value match with at least one other instruction within specific instruction window lengths. The degree of similarity between operand values further increases when we relax the constraint of exact matches to approximate matches, with the number of bit differences (Hamming Distance [17]) as the distance metric. In order to study value similarity, we used Intel Pin [18] to generate single-thread traces annotated with each instruction's source operand values, and pass them into a self-developed Python post-processor to find values that are similar with those in nearby instructions.

While both benchmark suites show value locality from program-related and input-related patterns, EdgeBench shows more commonality among inherently correlated data inputs compared to SPEC2017, which increases the percentage of value locality for small but non-zero hamming distances. If the microarchitecture can locate these similar values, it can effectively decrease bit-level switching activities on critical datapaths including ALUs and downstream execution units.

III. Implementation Insights

A. Predicting Value Locality

An ideal value-aware scheduler would know the source operand values of instructions before making lane assignments, but obtaining these values is impractical. Although operand values are available in the pipeline when the instruction is ready to be issued, accessing them either requires additional reads from the register file or complex logic for reading from bypassing. Both of these alternatives add complexity and require extra pipeline stages on the critical path. Scheduling decisions need to be done by **predicting operand correlations before knowing the actual values**. To avoid adding latency, the prediction can be done **off the critical path** between the fetch and the issue stage, based on the instruction's PC that is available at the fetch stage.

B. Targeting Instructions

The proposed scheduler can focus on rescheduling *ready-to-execute* instructions. Ready-to-execute instructions are those with no unresolved dependencies, and can be issued immediately if an execution lane is available. The concepts for the scheduler can be applied to both non-memory and memory instructions. While non-memory instructions are more intuitive, reordering memory instructions requires meticulous management for memory dependencies.

C. Maintaining Original Performance

To preserve original performance of the application, the scheduler could be made to only reassign instructions launched in the same cycle to difference execution lanes for better value matches. While this approach can bring decent energy reduction, we can further reorder instructions in different cycles to exploit value locality beyond neighboring cycles with minimal performance impact. However, for instructions without immediate value correlation matches, indefinitely delaying issuing to wait for future matches might lead to performance drop. The impact is serious if the instructions on the critical path are being pushed back when there is no value match. To preserve original performance, we need to prioritize issuing instructions on the critical path.

IV. Broader Implications

The principles for designing value-aware schedulers can be applied to value locality on other granularity levels, other hardware platforms, and diverse scenarios for architectural improvements. Exploring the unique value locality characteristics of specialized operand types, such as Intel's X86-64 architecture's Vector operands, or specific instructions types, can lead to fine-tuning for different operands or instructions for energy reduction. The scheduling strategy can also be applied to architectures beyond the CPU we modeled. The approach is feasible for any system with flexible instruction issuing.

Acknowledgment

We thank the members of the Princeton Parallel Group for their insightful discussions. This material is based upon work supported by the National Science Foundation under Award No. CCF-1822949 and Award No. CNS-1823222. This material is based upon work supported by a Princeton Andlinger Center Innovation Award and a Princeton SEAS Innovation Award. This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] S. Kaxiras and M. Martonosi, *Computer architecture techniques for power-efficiency*. Morgan & Claypool Publishers, 2008.

[2] B. Calder, P. Feller, and A. Eustace, "Value profiling," in *Proceedings of 30th Annual International Symposium on Microarchitecture*. IEEE, 1997, pp. 259–269.

[3] S. Wen, M. Chabbi, and X. Liu, "Redspy: Exploring value locality in software," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 47–61.

[4] B. Calder, P. Feller, and A. Eustace, "Value profiling and optimization," *Journal of Instruction-Level Parallelism*, 1999.

[5] I. R. de Assis Costa, H. N. Santos, P. R. Alves, and F. M. Q. Pereira, "Just-in-time value specialization," *Computer Languages, Systems & Structures*, vol. 40, no. 2, pp. 37–52, 2014.

[6] K. Zhou, Y. Hao, J. Mellor-Crummey, X. Meng, and X. Liu, "Gvprof: A value profiler for gpu-based clusters," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[7] M. N. Wegman and F. K. Zadeck, "Constant propagation with conditional branches," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 13, no. 2, pp. 181–210, 1991.

[8] D. Callahan, K. D. Cooper, K. Kennedy, and L. Torczon, "Interprocedural constant propagation," in *Proceedings of the 1986 SIGPLAN symposium on Compiler construction*, 1986, pp. 152–161.

[9] P. Su, S. Wen, H. Yang, M. Chabbi, and X. Liu, "Redundant loads: A software inefficiency indicator," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 982–993.

[10] X. You, H. Yang, Z. Luan, D. Qian, and X. Liu, "Zerospy: exploring software inefficiency with redundant zeros," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14.

[11] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[12] C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cérin, and J. Wan, "Energy aware edge computing: A survey," *Computer Communications*, vol. 151, pp. 556–580, 2020.

[13] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, p. 171–209, apr 2014. [Online]. Available: https://doi.org/10.1007/s11036-013-0489-0

[14] M. Gheisari, G. Wang, and M. Z. A. Bhuiyan, "A survey on deep learning in big data," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 2, 2017, pp. 173–180.

[15] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–42. [Online]. Available: https://doi.org/10.1145/3185768.3185771

[16] A. Das, S. Patterson, and M. Wittie, "Edgebench: Benchmarking edge computing platforms," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 175–180.

[17] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[18] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 190–200. [Online]. Available: https://doi.org/10.1145/1065010.1065034