

Alma Mater Studiorum Università di Bologna  
Archivio istituzionale della ricerca

A Hybrid Instruction Prefetching Mechanism for Ultra Low-Power Multicore Clusters

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

*Published Version:*

Payami, M., Azarkhish, E., Loi, I., Benini, L. (2017). A Hybrid Instruction Prefetching Mechanism for Ultra Low-Power Multicore Clusters. IEEE EMBEDDED SYSTEMS LETTERS, 9(4), 125-128 [10.1109/LES.2017.2707978].

*Availability:*

This version is available at: <https://hdl.handle.net/11585/624042> since: 2018-09-26

*Published:*

DOI: <http://doi.org/10.1109/LES.2017.2707978>

*Terms of use:*

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>).  
When citing, please refer to the published version.

(Article begins on next page)

This is the post peer-review accepted manuscript of:

M. Payami, E. Azarkhish, I. Loi and L. Benini, "A Hybrid Instruction Prefetching Mechanism for Ultra Low-Power Multicore Clusters," in IEEE Embedded Systems Letters, vol. 9, no. 4, pp. 125-128, Dec. 2017. doi: 10.1109/LES.2017.2707978

The published version is available online at: <https://doi.org/10.1109/LES.2017.2707978>

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

# A Hybrid Instruction Prefetching Mechanism for Ultra Low-Power Multi-core Clusters

Maryam Payami, Erfan Azarkhish, Igor Loi, and Luca Benini, *Fellow, IEEE*

**Abstract**—The instruction memory hierarchy plays a critical role in performance and energy efficiency of ultra-low-power processors for the Internet-of-Things (IoT) end-nodes. This is mainly due to the extremely tight power envelope and area budgets, which imply small instruction-caches (I-Cache) operating at very low supply voltages (near-threshold). The challenge is aggravated by the fact that multiple processors, fetching in parallel, require plenty of bandwidth from the I-Caches. In this paper, we propose a low-cost and energy efficient hybrid instruction-prefetching mechanism to be integrated with an Ultra-Low-Power (ULP) multi-core cluster. We study its performance for a wide range of IoT applications, from cryptography to computer vision, and show that it can effectively improve the hit-rate of almost all of them to above 95% (average performance improvement of over  $2\times$ ). In addition, we designed our prefetcher and integrated it in a 4-cores cluster in 28nm FDSOI technology. We show that system's power consumption increases only by about 11% and silicon area by less than 1%. Altogether, a total energy reduction of  $1.9\times$  is achieved, thanks to more than  $2\times$  performance improvement, enabling a significantly longer battery life.

**Index Terms**—Ultra Low-power Embedded Multi-cores, Energy Efficiency, Instruction Cache, Instruction Prefetching

## I. INTRODUCTION

Prefetching is an extensively studied technique for data-memory latency hiding in high-performance processors. Yet, instruction prefetching is not as critical in such systems for two reasons. First, the data working-set in large-scale applications is much larger than the code working-set. Second, the code is cache-friendly, so a moderately sized instruction-cache (I-Cache) achieves very high hit-rates, and architects rightfully focus on the main bottleneck which is data caching [1]. In parallel ULP processors for the Internet-of-Things (IoT) end-nodes, the situation is reverted: data working sets are smaller and highly amenable to tiling, double buffering, and DMA management, since most workloads are linked to near-sensor data processing [2][3]. On the other hand, the on-chip area budget is much tighter in IoT end-nodes. This squeezes the size of the I-Caches, also because they are often implemented with low-density memories which can operate at low voltages (e.g. standard-cell based caches [4][5]). This area squeeze is further exacerbated by the fact that there are multiple processors fetching in parallel, so the instruction memory system must deliver plenty of bandwidth, requiring complex

logic such as L0-buffers [6] and broadcast-interconnect [5], which further increase the area dedicated to the instruction memory. Therefore, in this context it is highly valuable in term of area and energy-efficiency to keep the I-Cache small, while at the same time exploit the latency-hiding opportunities offered by prefetching to reduce the impact of cache-misses [7]. Another factor is that the speed gap with the background on-chip L2-memory (dense SRAM or even embedded flash), where the entire code is stored, is not so huge, so the amount of prefetched information can be kept small (a direct consequence of Little's law [2][5]). This also works in the direction of making instruction prefetching highly effective.

Looking at the state-of-the-art instruction prefetching techniques, a simple yet effective method is called next-line prefetching (NLP) [1]. In this scheme, when a cache line is fetched, a prefetch for the next sequential line(s) is also initiated. One way to do this is called prefetch-on-miss. NLP exploits spatial locality for sequential access patterns, but for conditional/unconditional branches and function calls it is ineffective [8]. The Stream prefetcher (STP), similarly, follows a pattern or rule. As long as it persists, it issues a stream of prefetch requests [8]. Target-line [1] tries to prefetch non-sequential cache lines by maintaining a target prefetch table in hardware. Combining this method with NLP can take advantage of both mechanisms for sequential/non-sequential code. The main disadvantage of this method is a significant hardware cost for the table and the associated logic to perform the table lookups and updates. Also, the effective-address of the control-instructions need to be known even before their execution. This can be very costly in terms of logic delay and area. Plus, the compulsory misses do not benefit from it [1].

A Markov Prefetcher [8] monitors the address stream and builds a Markov prediction table in hardware. The learning phase of this scheme is usually long and its hardware cost is significant. A wrong-path prefetcher combines the target and next-line mechanisms, with the major difference that no target-line addresses are saved and no attempt is made to prefetch only the correct execution path [1]. Instead, in the simplest case both paths of conditional branches are always prefetched. This method can perform a potentially useful prefetch for a branch which is not taken, if the execution returns to it in the near future and it is then taken. No extra hardware is required above NLP, but cache pollution and the large amount of extra traffic are problematic.

Lastly, software-prefetching (SWP) [1] is based on the use of some form of explicit fetch instructions with very little added hardware. The only difficulty lies in correct placement of the fetch instructions in the code (also known as prefetch scheduling): It is possible to gain significant advantages by inserting a few fetch instructions manually in strategic portions

M. Payami, E. Azarkhish, and I. Loi are with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy (e-mails: maryam.payami@studio.unibo.it, {erfan.azarkhish, igor.loi}@unibo.it).

L. Benini is with the Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology Zurich, 8092 Zurich, Switzerland, and also with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy (e-mail: lbenini@iis.ee.ethz.ch).

This project has received funding from the European Unions Horizon 2020 research and innovation program for the Oprecomp project (GA No. 732631).







the benchmarks. Moreover, NLP was enabled to aid SWP with prefetching the sequential code sections. This way both the original sequential misses and most of the additional misses due to the code size increase can be eliminated. Please note that the manually inserted SWP instructions only increase the size of the sequential blocks, and these blocks are perfectly prefetched by NLP. More than 91% hit-rate was achieved with total execution only 8% higher than a 16 KB warm cache with 100% hits. This is interesting given that the area of the 16 KB cache is approximately  $16\times$  larger ( $2.8\times$  larger cluster).

Effect of the hybrid prefetcher on silicon area and power consumption is illustrated in Figure 3. As can be seen, the cluster area is increased by less than 1% (mainly due to the FSM and the additional read port to the tag array), and the total power consumption increase in the system (including the L2 memory and the peripherals) is around 11%, on the average. The power increase mostly occurs in the processors and the L2 memory, because of fewer stall cycles and the increased L2 traffic. Total energy reduction and performance improvement are shown in Figure 4, for each individual benchmark at its best prefetching configuration. On the average, energy-efficiency is improved by  $1.9\times$ , thanks to the average improvement of  $2.1\times$  in performance and very marginal increase in power consumption. This enables a significantly longer battery-life in energy constrained systems. Note that enlarging the cache is not as energy- and area-efficient as prefetching. An 8 KB cache without prefetching achieves similar performance as a 1 KB cache with prefetching, with 29% more power consumption and  $2\times$  larger cluster area. Also, further increase in cache size (to 2 KB) in presence of the prefetcher was found unnecessary, as performance only improves by less than 5% with more than 10% increase in total power (See Figure 3c). Another observation is that if we unroll *strassen*, its baseline performance will improve by  $2.3\times$ , and if we use prefetching with this unrolled version, another 35% improvement is achievable (less than 10% power consumption increase). This highlights that even though unrolling increases the code size it has a very positive impact on compiler scheduling, and if augmented with a proper prefetching mechanism it can be very beneficial for energy efficiency. To sum up, our proposed scheme allows simple in-order processors to issue multiple outstanding instruction fetch transactions towards the L2 memory, and provides them with latency hiding capabilities, similar to out-of-order processors with large issue widths. This way, stall-cycles are reduced and energy efficiency is improved to a great extent [10].

#### IV. CONCLUSIONS

In this paper we proposed a low-cost hybrid instruction prefetching mechanism based on SWP, NLP, and STP to be integrated with ultra low-power multi-core platforms with simple in-order cores. We studied a wide range of applications and grouped them into two categories: the ones with large computation loops, and the ones with many function calls. We showed that for most of the benchmarks in the first group a cooperation between NLP and STP can improve the ICache hit-rate to over 95% with an average performance improvement of over  $2\times$ . While for the second group, SWP+NLP can be effective, leading to a similar improvement. By synthesizing our designs using the state-of-the-art technologies we showed

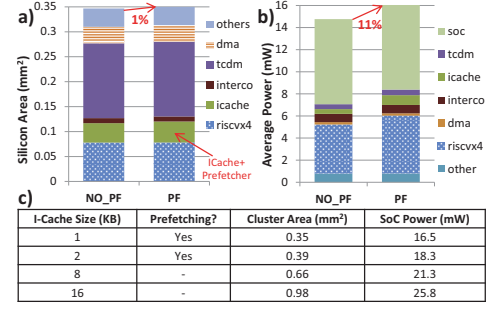


Fig. 3. (a) Silicon area ( $mm^2$ ) of the cluster (b) system power consumption, breakdown with/without the prefetcher. (c) Synthesis area and power consumption for cycle time of 2 ns.

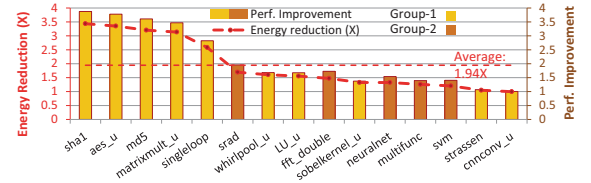


Fig. 4. Total energy reduction (left-axis) and performance improvement (right-axis) due to prefetching, sorted in decreasing order of energy reduction.

that the increase in system's power (about 11%) and cluster's silicon area (less than 1%) are negligible. Overall, our proposed prefetching scheme allows for an average energy reduction of  $1.9\times$  over the range of studied applications. The future directions include studying the effectiveness of the proposed schemes for larger (IoT gateway) system configurations, and complex parallel workloads including operating systems and virtual memory. We will also study compiler modifications to automatically insert software prefetch commands in the code.

#### REFERENCES

- [1] B. Falsafi and T. F. Wenisch, *A Primer on Hardware Prefetching*, M. Martonosi, Ed. Morgan & Claypool, 2014.
- [2] S. Pei, M. S. Kim, and J. L. Gaudiot, "Extending amdahl's law for heterogeneous multicore processor with consideration of the overhead of data preparation," *IEEE ESL*, vol. 8, no. 1, pp. 26–29, March 2016.
- [3] D. Rossi, F. Conti, A. Marongiu *et al.*, "PULP: A parallel ultra low power platform for next generation IoT applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, Aug 2015, pp. 1–39.
- [4] A. Teman, D. Rossi, P. Meinerzhagen *et al.*, "Power, area, and performance optimization of standard cell memory arrays through controlled placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 4, pp. 59:1–59:25, May 2016.
- [5] I. Loi, D. Rossi, G. Haugou *et al.*, "Exploring multi-banked shared-L1 program cache on ultra-low power, tightly coupled processor clusters," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15, 2015, pp. 64:1–64:8.
- [6] M. Gautschi, P. D. Schiavone, A. Traber *et al.*, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on VLSI Systems*, vol. PP, no. 99, pp. 1–14, 2017.
- [7] A. Panda and K. S. Chatha, "An embedded architecture for energy-efficient stream computing," *IEEE ESL*, vol. 6, no. 3, pp. 57–60, 2014.
- [8] S. Mittal, "A survey of recent prefetching techniques for processor caches," *ACM Comput. Surv.*, vol. 49, no. 2, pp. 35:1–35:35, Aug. 2016.
- [9] J. Lee, H. Kim, and R. Vuduc, "When prefetching works, when it doesn't, and why," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 1, pp. 2:1–2:29, Mar. 2012.
- [10] M. Payami, "Instruction prefetching techniques for ultra low-power multicore architectures," Master's thesis, Univ. of Bologna, Bologna, 2016.
- [11] J. Pallister, S. J. Hollis, and J. Bennett, "Identifying compiler options to minimize energy consumption for embedded platforms," *The Computer Journal*, vol. 58, no. 1, pp. 95–109, 2015.